Signature Neural Networks: Definition and Application to Multidimensional Sorting Problems

Roberto Latorre, Francisco de Borja Rodríguez and Pablo Varona

Abstract—In this paper we present a self-organizing neural network paradigm that is able to discriminate information locally using a strategy for information coding and processing inspired in recent findings in living neural systems. The proposed neural network uses (i) neural signatures to identify each unit in the network, (ii) local discrimination of input information during the processing, and (iii) a multicoding mechanism for information propagation regarding the who and the what of the information. The local discrimination implies a distinct processing as a function of the neural signature recognition and a local transient memory. In the context of artificial neural networks none of these mechanisms has been analyzed in detail, and our goal is to demonstrate that they can be used to efficiently solve some specific problems. To illustrate the proposed paradigm, we apply it to the problem of multidimensional sorting, which can take advantage of the local information discrimination. In particular, we compare the results of this new approach with traditional methods to solve jigsaw puzzles and we analyze the situations where the new paradigm improves the performance.

Index Terms—Neural signatures, Multicoding, Local discrimination, Local contextualization, Self-organization, Jigsaw puzzles.

I. INTRODUCTION

Recent experiments in living neural circuits known as central pattern generators (CPG) show that some individual cells have neural signatures that consist of neuron specific spike timings in their bursting activity [34], [33]. Model simulations indicate that neural signatures that identify each cell can play a functional role in the activity of CPG circuits [22], [23], [24]. Neural signatures coexist with the information encoded in the slow wave rhythm of the CPG. Readers of the signal emitted by the CPG can take advantage of these multiple simultaneous codes and process them one by one, or simultaneously in order to perform different tasks [23]. The who and the what of the signals can be used to discriminate the information received by a neuron by distinctly processing the input as a function of these multiple codes. These results emphasize the importance of cell diversity for some living neural networks and suggest that local discrimination is important in systems where neural signatures are present. This kind of information processing can be a powerful strategy for neural systems to enhance their capacity and performance.

Artificial neural networks (ANN) are inspired to some extent from their biological counterparts. However, in the context of artificial neural computation, phenomena such as local recognition, discrimination of input signals and multicoding strategies have not been analyzed in detail. Most traditional ANN paradigms consider network elements as indistinguishable units, with the same transfer functions, and without mechanisms of transient memory in each cell. None of the existing ANN paradigms discriminates information as a function of the recognition of the emitter unit. While neuron uniformity facilitates the mathematical formalism of classical paradigms [13], [1], [36], [3], [17] (which has largely contributed to their success [39]), some specific problems could benefit from other approaches.

Here we propose a neural network paradigm that makes use of neural signatures to identify each unit of the network, and multiple simultaneous codes to discriminate the information received by a cell. The network self-organization is based on the signature recognition and on a distinct processing of input information as a function of a local transient memory in each cell that we have called the local informational context of the unit. The efficiency of the network depends on a trade-off between the advantages provided by the local information discrimination and its computational cost.

In this paper we discuss the application of SNNs to solve multidimensional sorting problems. In particular, to fully illustrate the use of this neural network and to evaluate its performance, we apply this formalism to the task of solving canonical jigsaw puzzles.

The paper is organized as follows. In section II we present the general formalization of the proposed paradigm. In section III, we (i) discuss its application to generic multidimensional sorting; and (ii) provide an implementation for this kind of problems. To test the performance, in section IV we (i) review the jigsaw puzzle problem and the traditional algorithms to solve it; (ii) provide a specific solution using a signature neural network; (iii) describe the methods to evaluate the performance; and (iv) we present our quantitative results on the comparison of this new approach with traditional methods to solve jigsaw puzzles and we analyze the situations where the new paradigm improves the performance (section IV-H). Finally, in the Appendix we illustrate in detail the evolution of the network with another example of multidimensional sorting.

II. SIGNATURE NEURAL NETWORK FORMALIZATION

In this section we present the signature neural network (SNN) paradigm. Behind this new paradigm, there are four main ideas:

- Each neuron of the network has a signature that allows its unequivocal identification by the rest of the cells.
• The neuron outputs are signed with the neural signature. Therefore, there are multiple codes in a message (multicoding) regarding the who and the what of the information.
• The single neuron discriminates the input signals as a function of (i) the recognition of the emitter signature and (ii) a transient memory that keeps track of the information and its sources. This memory provides a contextualization mechanism to the single neuron processing.
• The network self-organization relies to a large extent on the local discrimination by each unit.

A. Signature Neural Network Definitions

The formalism requires the definition of several terms that will be used in the following sections. Some of the signature neural network definitions are open and depend on the specific problem to be solved. This provides a framework that can be applied to different problems by only customizing the open definitions. To illustrate the use of the SNN, we will fix these open definitions for the general multidimensional sorting problem and the particular case of the jigsaw puzzle solver in sections III-A and IV-D, respectively.

• Neuron or cell: the processing unit of the network.
• Neuron signature: the neuron ID in the network. This ID is used for the local information discrimination.
• Neuron data: information stored in each neuron about the problem.
• Neuron information: the joint information about the who (neuron signature) and the what (neuron data) of the cell.
• Synapse: connection between two neurons.
• Neuron neighborhood: cells directly connected to the neuron. This concept is used to define the output channels of each neuron. The neuron neighborhood can change during the evolution of the SNN.
• Local informational context: transient memory of each neuron to keep track of the information and its sources. This memory consists of a subset of neuron informations from other cells received in previous iterations. The maximum size of the context ($N_{\text{context}}$) is the maximum number of elements in this set, and it is an important parameter of the algorithm. The neuron signature and the local informational context are the key concepts of the SNN.
• Local discrimination: the distinct processing of a unit as a function of the recognition of the emitter and the local informational context.
• Message: the output or total information transmitted through a synapse between two neurons in a single iteration. The message consists of the neuron information of a subset of cells that are part of the context of the emitter plus its own neuron information (see below). The maximum message size is equal to $N_{\text{context}}$. The input to a neuron consists of all messages received at a given iteration.
• A receptor starts recognizing the signature of an emitter cell during the message processing when it detects that the neuron data of the emitter is relevant to solve the problem (emitter and receptor data are compatible). The network self-organization is based on this recognition. The meaning of “relevant” depends on the specific problem.
• Information propagation mode: depending on the problem, the information propagation can be monosynaptic or multisynaptic. Monosynaptic means that each neuron can receive only one input message per iteration. The information propagation is bidirectional between cells.
• A neuron belongs to a cluster if it recognizes the signature of all the neurons in its neighborhood. The clusters allow to simplify the processing rules of the SNN.

B. Algorithm

The connectivity, the neuron data and the local informational contexts of all the network units are previously initialized. Depending on the problem, connectivity and neuron data initialization can be random or heuristic. Three different context initializations can be considered: (i) A free context initialization where initially the context of every neuron is empty. In this way the cells have no information about the rest of the network. (ii) A random context initialization where the context of the neurons is chosen randomly. And (iii) a neighborhood context initialization where all the contexts are coherent with the neighborhood of each neuron.

After the initialization, the algorithm consists in the iteration of three different steps for each neuron in the network until the stop condition is fulfilled. Note that the network self-organization takes place both in step 1 and 3 by modifying the network connections.

1 Process synaptic inputs

In this phase of the algorithm each neuron applies the local information discrimination:

• First, the cell discriminates the input messages as a function of the emitter signature to determine which of them will pass to a second discrimination stage. If no signatures are recognized (a situation likely in the first iterations), all messages pass to the second stage.
• Second, the neuron uses the memory of the processing in previous iterations stored in its local informational context to select the set of neuron informations from the messages that will be finally processed.
• Third, the cell processes these set of neuron informations by applying its corresponding transfer functions or processing rules (which are specific of the problem to solve). If the neuron data processed is relevant to solve the problem, the cell starts recognizing the corresponding signature and establishes a new connection with the cell identified by this signature.
• Finally, as the last step of this phase, the local informational context of the receptor is updated using the neuron information set analyzed during the processing of the input messages.
Local discrimination can lead to changes in the network connectivity. Network reconfiguration as a function of the local discrimination implies a nonsupervised synaptic learning. Clusters represent partial solutions to the problem. Neurons belonging to a cluster have the same processing rules.

2 Propagate information
During this phase neurons build and send the output messages. For this task each neuron adds its own information to the local informational context and signs the message. If the message size reaches the $N_{\text{context}}$ value, the neuron information from the oldest cell of the context is deleted from the message (this will be illustrated in Fig. 4 for the puzzle solver case). The output message of a neuron is the same for all output channels.

3 Restore neighborhood
If a neuron has not reached its maximum number of neighbors, it randomly tries to connect to another neuron in the same situation (only one connection per neuron and iteration). First, it tries to connect to neurons from its local informational context and, if this is not possible, to other cells. This allows to maximize the information propagation in the network. To establish synapses with cells not belonging to the local context allows propagating information to other regions of the network.

### III. SNNs for Multidimensional Sorting

An ANN paradigm based on local discrimination relies on the criteria used to perform discrimination, which necessarily depends on the problem to be solved. This implies that an SNN must be designed thinking on the specific problem at hand. To illustrate the concept and the applicability of the SNN paradigm, we will apply it to the problem of multidimensional sorting. The ideas relating neural signatures with local information discrimination have a direct application in the wide scope of multidimensional sorting problems. This is an example in which a solver can take advantage of local information discrimination of SNNs can lead to an efficient solution of multidimensional sorting problems with local order criteria.

#### A. Customization of the signature neural network

In this section we describe how to use a SNN to solve a general multidimensional sorting problem. For this task, we have to fix some of the open definitions, conditions and constraints for the algorithm.

All definitions of section II-A apply to build the SNN network. However, the parameters regarding the dimension of the problem, the number of neighbors, the final structure of the network, and most of all, the recognition for the local discrimination task, will depend on the specific problem at hand.

Common grounds for all multidimensional sorting problems to be solved with a SNN are:

- The number of neurons of the network is equal to the number of elements to sort. There is a one to one relationship between the neurons and the elements to sort.
- The neuron signature can be the neuron number or some other value that allows to identify unequivocally each neuron.
- The neuron data of each cell is a structure with information about the element to sort in each dimension (for example, in a scheduling problem neuron data will be a task with a cost, effort, priority, etc).
- The network is $d$-dimensional, where $d$ is the number of dimensions of the problem (each dimension can use a specific sorting algorithm, global or local).
- The information propagation mode is multisynaptic.
- The compatibility for an element is given by the sorting criteria. If the sorting criteria is local, elements can only be compatible or not compatible. If the sorting criteria is global, a best compatibility measure can be assigned among different elements. During the algorithm evolution, neurons can change dynamically their compatibilities and, thus, the set of signatures recognized in a given iteration to adjust the discrimination rules.
- The sorting criteria defines two possible neighbors in each dimension, the previous and the next element to be sorted.
- If none of the neurons has learned a new signature for a given amount of iterations, the network reaches the stop condition.

#### B. Implementation of the signature neural network

Here we present a brief pseudocode for the SNN paradigm to solve a general multidimensional sorting problem. The following notation is used: $expression \rightarrow variable$ means that variable takes the value of the evaluation of the expression; $variable[]$ means that variable is a vector; and $Signature(n_i)$, $Compatibility(n_i, n_j)$ and $Context(n_i)$ denote the signature, the compatibility with $n_j$, and the local informational context of neuron $n_i$, respectively. $P$ is the probability to establish a new connection with a cell of its local informational context during the processing phase. $T$ is the threshold for the maximum number of iterations in which a neuron is allowed to have an incomplete neighborhood.

The initialization on the network consists of following steps:

1. Assign each element to sort to a neuron of the network.
2 Establish random connections between cells to build the initial network architecture.

3 For each neuron of the network → \( n_i \):
   3.1 Initialize Context\((n_i)\) using one of the initializing algorithms (see section II-B).

   The neural network main function is repeated until the end condition is fulfilled. This function consists of the following steps in each neuron:

   1 Process synaptic input:
      1.1 Synaptic input messages → \( inputs[] \)

      Regarding the who of the information (steps 1.2 and 1.3 constitute the first discrimination stage, while step 1.5 corresponds to the second discrimination stage):

      1.2 Select from \( inputs[] \) those messages sent by an emitter with a recognized signature → \( \text{recognized}[] \)

      1.3 If \( \text{recognized}[] \) is empty, select all messages from \( inputs[] \) → \( \text{recognized}[] \)

      1.4 For each emitter in \( \text{recognized}[] \) → \( \text{emitter} \)
         * Search for a side of \( n_i \) without a neighbor → \( \text{empty} \).
         * Search for a neuron in Context\((n_i)\) without a neighbor in the opposite side to \( \text{empty} \) → \( n_j \).
         * If \( n_j \) exists, connect \( n_i \) and \( n_j \) through \( \text{empty} \).
         * Else, if \( n_i \) has had incomplete neighborhood for a number of iterations larger than \( T \):
            - Choose randomly among the non-neighbors of \( n_i \) a neuron different than \( n_i \) → \( n_k \).
            - Break the connection of \( n_k \) in the opposite side to \( \text{empty} \) and connect \( n_i \) and \( n_k \) through \( \text{empty} \).

   1.5 Select randomly \( N_{\text{context}} \) neurons not including in Context\((\text{receptor})\) from messages in \( \text{recognized}[] \) → \( \text{in} \)

      Regarding the what of the information:

      1.6 For each dimension, process information of \( \text{in} \):
         * Sort neurons of \( \text{in} \) with the corresponding sorting algorithm → \( \text{sorted}[] \)
         * Choose from \( \text{sorted}[] \) those neurons that are compatible with \( \text{receptor} \) → \( n_i \)
         * If \( n_i \) exists
            - Choose the corresponding neighbor of \( \text{receptor} \) for the corresponding dimension → \( \text{neighbor} \)
            - If \( \text{Compatibility}(\text{receptor}, n_i) \) is better than \( \text{Compatibility}(\text{receptor}, \text{neighbor}) \):
              (i) Break connection between \( \text{receptor} \) and \( \text{neighbor} \) and connect \( \text{receptor} \) and \( n_i \); (ii) \( \text{receptor} \) starts recognizing Signature\((n_i)\); (iii) \( \text{receptor} \) stops recognizing Signature\((\text{neighbor})\).
         * Else search in Context\((\text{receptor})\) for a cell with incomplete neighborhood → \( n_j \)
            - If \( n_j \) exists, connect \( \text{receptor} \) and \( n_j \) with probability \( P \).
      1.7 Update Context\((\text{receptor})\) with \( \text{in} \).

2 Propagate information:

2.1 For each neuron of the network → \( n_i \):
   * For each neighbor of \( n_i \) → \( n_j \), send messages between \( n_i \) and \( n_j \).

3 Restore neighborhood:

3.1 For each neuron with incomplete neighborhood → \( n_i \):
   * Search for a side of \( n_i \) without a neighbor → \( \text{empty} \).
   * Search for a neuron in Context\((n_i)\) without a neighbor in the opposite side to \( \text{empty} \) → \( n_j \).
   * If \( n_j \) exists, connect \( n_i \) and \( n_j \) through \( \text{empty} \).
   * Else, if \( n_i \) has had incomplete neighborhood for a number of iterations larger than \( T \):
      - Choose randomly among the non-neighbors of \( n_i \) a neuron different than \( n_i \) → \( n_k \).
      - Break the connection of \( n_k \) in the opposite side to \( \text{empty} \) and connect \( n_i \) and \( n_k \) through \( \text{empty} \).

This general implementation can be used in a wide variety of problems by customizing the discrimination rules. In the Appendix, we describe in detail a multidimensional sorting example that helps the reader to further understand the use of the local informational context and the local information discrimination. To test the performance of the SNN framework, we will first consider another example in which the discrimination rules are well known.

IV. SNN FOR THE JIGSAW PUZZLE PROBLEM

A. Problem definition

Jigsaw puzzles are a specific case of multidimensional sorting problems in which the order criteria is local and it is given by the fitting among pieces. A typical jigsaw puzzle is a two-dimensional picture that has to be rebuilt from different fragments or pieces. Once the pieces are mixed, the solution to the problem consists in reassembling them into the original picture. The difficulty of solving the puzzle depends mainly on the number of pieces, on their assembly complexity and on the graphical representation of the picture. To rebuild a jigsaw puzzle without the original image is a NP-complete problem [11].

To efficiently solve jigsaw puzzles is considered a classical fitting or pattern recognition problem and the algorithms to solve it may have potential applications in many different fields of knowledge such as archeology, art restoration, failure analysis, steganography and others. For example, such algorithms have been used to reassemble manuscripts from separate pieces [26], to rebuild broken objects [37], [20], [25], [40] to send secure messages in a non secure channel [45], to hide secret messages in seemingly innocuous carriers [12] or even to design evolutionary algorithms to solve complex problems [44].

Although some of these problems can be considered as 3D jigsaw puzzle assembling, we focus our work on solving 2D puzzles as the one shown in Fig. 1. Jigsaw puzzle pieces are typically rectangular and they fit with their contiguous neighbors, which are usually four except for the border pieces. The full picture is usually square-shaped. Puzzles with these constraints are called canonical jigsaw puzzles [43]. Our method and results can be easily extended to rebuild 3D objects from fragments with a different number of neighbors.

Although the solution to the problem involves several different tasks, the research literature about jigsaw puzzles and reconstruction of broken objects is mainly focused on algorithms
to test the matching of pieces according to their shape [14], [30], [27], [4], [42], [6], [38], [15] and, more recently, also on image (texture and color) matching [31], [21], [8], [43]. Different techniques have been used for this goal: shape matching [43], image merging [43], neural networks [32], genetic algorithms [35], best first search [5], etc. To solve the jigsaw puzzle, pairs of pieces are chosen (randomly or with a heuristic method) to test their fitting. Several tasks related to solving the puzzle can also be considered to be part of a sorting or classification problem, in the sense that pieces must be sorted and clustered in different groups to reduce the space search for a correct fitting. However, the performance of the associated sorting algorithm is usually disregarded. Typically it is thought that the efficiency to solve the problem is mainly related to the way the solver determines if two pieces can fit, rather than to the way the pieces are sorted and classified to test this fitting. Here we focus on the sorting and classification tasks.

The jigsaw puzzle problem is interesting in the context of our study because the sorting and classification algorithms are multidimensional sorting problems that can take advantage of local information discrimination to reduce the space search for correct fittings [28], [40]. If similar pieces are grouped into sets, each piece only needs to be compared with those in the same set. We have used the proposed paradigm to build a neural network that is able to efficiently implement the fitting algorithm. With this paradigm, we improve the performance to solve jigsaw puzzles by optimizing the strategy to choose pairs of pieces to test their fitting.

B. The general solver schema

Traditional puzzle solver algorithms follow a common general schema to find the correct solution. The reconstruction of the puzzle (or the object in a general case) is usually an exhaustive search over all pieces or fragments trying to find the best fittings. Therefore we can consider that the general algorithm is as follows:

- Choose a piece ($P_1$) from the set of available pieces.
- Search for one piece ($P_2$) that fits with $P_1$ through one of its borders.
- Assemble both pieces in a new single piece.
- Add this new piece to the set of available pieces, deleting $P_1$ and $P_2$.
- Back to the first step until only one piece is left.

Differences between existing approaches arise both from the algorithm used to test the matching of pieces $P_1$ and $P_2$, and from the one used to select which pieces are to be tested. Therefore, the performance of a jigsaw puzzle solver depends mainly on these two algorithms.

C. Traditional algorithms to choose pieces to compare

In classical approaches to solve jigsaw puzzles, the algorithms used to select a pair of pieces to test their fitting are based on the way humans solve jigsaw puzzles. Firstly they search for border pieces. Later they can choose to group the rest of pieces into different sets (e.g. according to the number of straight edges of each piece, according to their colors or any other similitude metrics) to make the search easier by focusing only on pieces with a greater probability of fitting. Finally, they try to find the correct fitting for each piece of the puzzle.

Traditional algorithms for piece selection are stochastic in different levels. In most cases they are brute force techniques that sort pieces until the correct solution is found [6]. Pieces are placed randomly and if the solution it is not reached there is a new random search iteration. Alternatively, each piece is compared with all the rest until the correct fittings are found [14]. In other cases, for each piece of the puzzle the fitting is tested only for a subset of the available pieces. For example, in many approaches key pieces are identified first and then assembled independently using different heuristics. This set of approaches sort all possible matchings according to specific measures to find the best candidates to fit as a function of the shape and/or graphical content of the piece. Here, pieces are chosen following an order (best first, highest confidence first, etc) and not randomly [42], [5], [20], [15], [43], [10], [28]. Thus, the space search is reduced. These algorithms require the calculation of complex similitude measures to be effective. The measures that are easy to calculate do not always give a good performance. Experiments reported in the literature using this kind of algorithms use puzzles with less than 300 pieces. A detailed comparison between different image feature solving methods can be found in [28].

D. Signature neural network to solve jigsaw puzzles

In this section we describe in detail how to use the SNN to solve the jigsaw puzzle problem, and in the next section we will provide a pseudocode for the implementation of this algorithm. The SNN paradigm defines a different search than the general puzzle-solver search schema described in section IV-B. Here the processing units are neurons that try to find the best fitting locally.

The SNN described in section III-B for the general multidimensional sorting case can be easily adapted to the jigsaw puzzle problem. However, to allow a fair comparison between

Fig. 1. Example of a canonical jigsaw puzzle with a picture of Mount Kilimanjaro. Pieces are rectangular and the number of neighbors is four except for the border pieces. Corners have two neighbors and the rest of border pieces have three. The solution to the puzzle consists in reassembling pieces into the original picture once they are mixed.
the performance of the SNN and a traditional stochastic algorithm, we need to impose some restrictions to the general framework. However, note that the SNN can also be applied without these restrictions as we will discuss later.

- The number of neurons of the network is equal to the number of pieces of the puzzle.
- The neuron signature is the neuron number. There exist different matching algorithms that use different metrics to represent the characteristics of a piece or fragment [19], [18], [41]. For example, objects can be represented by "shape signatures" which are strings that are obtained by an approximation of the boundary curve. The signature of a neuron could be the shape signature of the piece that it contains. However, as we are not interested in evaluating the fitting algorithm, to simplify our implementation we use the neuron number as the neural signature.
- Now, the neuron data of each cell is one piece of the puzzle (this is illustrated in Fig. 2).
- As we solve canonical jigsaw puzzles, the maximum number of neighbors is four, one for each side of the piece that the neuron represents (up, down, left and right). The neighbor order is important, up-down and left-right are opposite sides. In a more general case, for example to rebuild broken objects, there could exist more than four neighbors. The SNN has periodic boundary conditions.
- The initial structure of the network is two-dimensional with each cell connected to its four nearest neighbors.
- In our sample, the information propagation mode is monosynaptic, i.e. only one input message is processed per iteration. Fig. 4 shows the way messages are built and propagated with this choice of parameters.
- When two neurons contain pieces with a complementary border (borders that match correctly), they are compatible. For example, in Fig. 2, since neurons 18 and 25 contain complementary pieces, they are compatible and, recognize their signatures.

E. Puzzle solver implementation

Here we present a brief pseudocode for the SNN paradigm to solve jigsaw puzzles with the same notation as in section III-B. In the simulations discussed in this paper, values of $P$ and $T$ are 0.1 and 10, respectively:

1. Process synaptic input:
   1.1 Synaptic input message $\rightarrow inputs$
when information propagation is monosynaptic, *inputs* only contains one input message.

Regarding the who of the information:

1.2 Select from *inputs[]* those messages sent by an emitter with a recognized signature and not in its correct position → *recognized[]*

1.3 If *recognized[]* is empty, select randomly a message of *inputs[]* → *recognized[]*

1.4 For each emitter in *recognized[]* → emitter
   * If receptor has recognized Signature(emitter), reconfigure the network to move pieces to their correct position (see Fig. 3).

1.5 Select randomly *Ncontext* neurons not including in *Context(receptor)* from messages in *recognized[]* → in

Regarding the what of the information:

1.6 For each dimension, process information of *in*:
   * Search in *in* for a neuron whose signature has not been recognized by receptor but has a complementary piece to *Piece(receptor)* in the corresponding dimension → *n*
   Note that the part of the incoming messages about neurons whose signature is recognized is not processed.
   * If *n* exists:
     - Connect receptor and *n*
     - receptor starts recognizing Signature(*n*)
     This means that the emitter will recognize the signature of the emitter in the next iteration.
   * Else, search in *Context(receptor)* for a cell with incomplete neighborhood → *nj*.
     - If *nj* exists, connect receptor and *nj* with probability *P*.

1.7 Set *Context(receptor)* equal to the set of neuron informations of *in*. If receptor belongs to a cluster do not include the neuron information of any neuron in its neighborhood to build *Context(receptor)*.

2 Propagate piece information:

2.1 For each neuron of the network whose corresponding piece is not in its correct position → *ni*

   * Search for a neighbor of *ni* with a signature not recognized by *ni* which contains a complementary piece of *Piece(ni) → nj*.

   * If *nj* exists, send messages between *ni* and *nj*.
   * Else, choose randomly a neighbor of *ni* (note that information propagation is monosynaptic) → *nk*.
     - If *nk* exists, send messages between *ni* and *nk*.

3 Restore neighborhood:

   3.1 For each neuron whose corresponding piece is not in its correct position and with incomplete neighborhood → *ni*:
     * Search for a side of *ni* without a neighbor → empty.
     * Search for a neuron in *Context(ni)* without a neighbor in the opposite side to empty → *nj*.
     * If *nj* exists, connect *ni* and *nj* through empty.
     * Else, if *ni* has had incomplete neighborhood for a number of iterations larger than *T*:
       - Choose randomly among the non-neighbors of *ni* a neuron different than *ni* → *nk*.
       - Break the connection of *nk* in the opposite side to empty and connect *ni* and *nk* through empty.

Note that the only significant change to the pseudocode described in section III-B is related to the what of the information.

**F. Methodology and validation**

1) *How to evaluate the performance of the signature neural network?*: To evaluate the SNN to solve jigsaw puzzles, we have compared its performance with the performance of a traditional stochastic algorithm (SA) based on the general solver schema described in section IV-B. The SA consists in the following steps:

- For each piece of the puzzle (*Pi*) repeat *N* times:
  - Choose randomly a piece of the puzzle (*Pj*)
  - If *Pi* and *Pj* have a complementary side:
    * Assemble both pieces in a new single piece
    * Add the new piece to the set of available pieces, deleting *Pi* and *Pj*.
- Back to first step until only one piece is left.

The number of attempts to find a complementary piece for each piece per iteration (*N*) is the main parameter of the SA. We consider this value equivalent to the context size in the SNN (*Ncontext*).

In section IV-C we reviewed the traditional algorithms to choose pieces to compare in order to solve the puzzles. All of
them are stochastic in different levels and can be described under this general scheme, some of them only by setting the number of attempts to find a complementary piece for each piece per iteration \((N)\). For the rest, specific rules for clustering pieces need to be added to execute the \(SA\) in different sets of pieces (for example, border pieces), or to use a priori knowledge about the matching of two pieces (e.g., their concavity and convexity). These set of rules can also be easily added to the \(SNN\). When they are applied, the improvement is equivalent for both methods under the same conditions. Here we do not use them to compare both approaches in the simplest case.

2) **What are the puzzles to solve?** To probe the viability of the proposed algorithm and to compare it with traditional approaches we have solved several puzzles of different sizes with the \(SNN\) and with the stochastic algorithm described in the previous section. In all our tests we have used computer generated canonical squared jigsaw puzzles of size \(n \times n\). To generate these puzzles, we divided pictures in \(n \times n\) squared fragments and mixed them randomly.

3) **How to test the piece matching?** To test the piece matching we use information about the overall picture. Before mixing pieces, we save the neighborhood of all the fragments. In this way, during the algorithm evolution, we can evaluate if two pieces fit or not.

4) **How to quantify the performance?** To assess the algorithm performance we use three measurements that will allow us to compare the different methods in terms of time requirements and effectiveness: the average number of iterations to solve the puzzles, the average total number of fitting tests needed, and the effective number of fitting tests (see below). These three measurements will allow to quantitatively analyze our results.

A method’s performance is often evaluated using the average time needed to solve the jigsaw puzzle. Let us define an iteration as a cycle of the algorithm in which all the processing units (pieces in the \(SA\) and neurons in the \(SNN\)) are updated. Therefore, this measure is equivalent for both methods in the sense that in each iteration they try to find the best fittings for all the pieces of the puzzle. To have a measure of the performance independent of the computer power and on the quality of the implementation, here we will quantify the performance of the algorithms in terms of the average number of iterations needed to solve puzzles of different sizes.

Fitting algorithms can be complex and computationally expensive. Therefore, performance is improved as the total number of fitting tests is reduced independently of the number of iterations. We consider that a fitting test takes place during the algorithm evolution every time the borders of two pieces are compared. For example, when reassembling two pieces of four borders a maximum of 16 fitting test are performed.

Finally, the effective number of fitting tests per iteration is defined as the percentage of correct matchings between pieces in each iteration of the algorithm. This quantity is used to assess the relationship between the two previous measurements.

To illustrate the results of the performance comparison between the proposed \(SNN\) and the \(SA\), we calculate the difference between the value of the above defined measures for each algorithm. Thus, let us define the following “distances”:

\[
\begin{align*}
    d_i &= \text{Iterations}_{SA} - \text{Iterations}_{SNN} \\
    d_{\text{tests}} &= \text{Tests}_{SA} - \text{Tests}_{SNN} \\
    d_{\text{eff}} &= \text{EffectiveTests}_{SNN} - \text{EffectiveTests}_{SA}
\end{align*}
\]

Negative values of these distances mean a poor performance by \(SNN\) as compared with the performance of the \(SA\). Note that the larger the number of iterations and the larger number of fitting tests, the worst is the performance. However, the larger the number of effective tests, the better the performance.

5) **Simulation parameter.** The main parameters in our simulations are the local informational context size (for the \(SNN\)) and the number of attempts to find complementary pieces in each iteration (for the \(SA\)). For each piece, these values indicate the maximum number of pieces for the fitting test per iteration. In this sense, we consider both parameters equivalent in order to compare the performance of the two algorithms. Here on these quantities are called “simulation parameter”.

In all our tests we set the simulation parameter to a percentage of the puzzle border length. For example, when we deal with puzzles of size \(50 \times 50\) and we say that the simulation parameter is \(10\%\), it means that the size of the local informational context of the \(SNN\) and the number of fitting attempts in the \(SA\) is equal to \(5\) (10\% of 50). Note that the storage requirement of the \(SNN\) is \(O(N \times N_{\text{context}})\), where \(N\) is the number of neurons in the network.

**G. Context initialization**

In order to test the dependency of the \(SNN\) with the initial conditions we have used the three different context initializations proposed in section II-B: (i) a free context initialization, (ii) a random context initialization, and (iii) a neighborhood context initialization.

**H. Results**

To assess the viability of the \(SNN\) paradigm we have solved several canonical jigsaw puzzles of different sizes: from puzzles of \(5 \times 5\) pieces to puzzles of \(100 \times 100\) pieces increasing the border size in steps of five pieces. In all cases we compare the results obtained with the \(SNN\) with those of solving the puzzles using the \(SA\) described in section IV-F1. As mentioned before, we evaluate the performance as a function of the simulation parameter: the size of the local informational context for the \(SNN\) and the number of attempts to find complementary pieces per iteration for the \(SA\). For each size, the simulation parameter goes from 10\% to 100\% in steps of 5\%.

The first result observed in our tests is that the \(SNN\) performance does not depend on the context initialization. There are only very small differences resulting from the three methods proposed to initialize the network. These differences are significant only when the size of the local informational context and the puzzle size are large (greater than 80\% and \(75 \times 75\), respectively). Taking into account this result, we
have decided to use the free context initialization in all the simulations discussed here, since this is the method that uses no a priori information to solve the problem.

We start the comparison between the performance of the SNN and the SA by analyzing the results of solving jigsaw puzzles in terms of the number of iterations (Fig. 5) and the number of fitting tests (Fig. 6) required to solve 100 different puzzles. For small puzzles, the performance of the neural network improves as the value of the simulation parameter increases, but it is never better than the performance of the SA. For large puzzles, the performance of the SNN is better than the SA for small values of the simulation parameter.

The extreme case is when the number of attempts to find a complementary piece is equal to the total number of pieces. In this case, puzzles can be solved only in one iteration. The signature neural network can never achieve this performance level because the SNN has an adaptation period to fill the local informational contexts of all the neurons with relevant information for each unit at the initial iterations. For example, with a simulation parameter equal to 100% (x-axis), the SA needs an average of 106 iterations, while the SNN needs an average of 219 (52% less performance). Thus, for large values of the simulation parameter, the SA requires always fewer iterations to solve the puzzle. However, for large puzzles, the computational cost of the local information discrimination is less significant compared with the total number of iterations needed to solve the puzzle. Therefore, the performance of the SNN becomes better as the context size is smaller.

Figure 5 shows the results for the evaluation of the mean number of iterations needed to solve 100 puzzles of 25 × 25 (top) and 100 × 100 pieces (bottom) as a function of the simulation parameter. This measure is smaller for the SA only for small puzzles with a small simulation parameter. In the rest of cases, the number of fitting tests needed to solve the puzzles is always less for the SNN.

Figure 6 shows the results for the evaluation of the mean number of fitting tests needed to solve 100 puzzles of 25 × 25 (top) and 100 × 100 pieces (bottom) as a function of the simulation parameter. This measure is smaller for the SA only for small puzzles with a small simulation parameter. In the rest of cases, the number of fitting tests needed to solve the puzzles is always less for the SNN.
approximately \(15 \cdot 10^7\) fitting tests, meaning an improvement of the performance by the SNN of 24%. These results suggest that the larger the puzzle, the better is the performance in terms of the number of fitting test of the SNN, independently of the simulation parameter.

To extend the analysis, we calculated the distances \(d_{it}\) and \(d_{test}\) to solve 100 different puzzles with the SA and the SNN for a wide range of puzzle sizes and simulation parameters (Fig. 7). The results are in agreement with those shown in Figures 5 and 6. For the mean number of iterations, the performance space can be divided in three different regions (Fig. 7, left panels). (i) For puzzles of moderate size (up to size \(50 \times 50\)) and a small simulation parameter (smaller than 20%), the performance of the SNN is not good compared with that of the SA. (ii) For puzzles with more than \(50 \times 50\) pieces with a simulation parameter between 10% and 30%, the performance of the SNN is better, i.e. the value of \(d_{it}\) is greater than 0. (iii) In the rest of cases, the performance of the SA is better, but very similar to that provided by the SNN. Regarding the number of fitting tests (right panels of Fig. 7), the performance space can also be divided in three different regions. (i) For small puzzles and small simulation parameters, the performance of the SA is better. (ii) The second region corresponds also to small puzzles, but now with the largest values for the simulation parameter. Here, the mean number of fitting test needed to solve the puzzles is very similar for both methods. (iii) For puzzles larger than \(45 \times 45\), the SNN has the best performance independently of the value of the simulation parameter. The improvement clearly increases with the puzzle size. For example, for puzzles of size \(75 \times 75\) the performance of the SNN improves approximately 10% with respect to the SA. For puzzles of \(100 \times 100\) pieces this improvement is 24%.

The results shown in Fig. 7 suggest that there is not a clear relationship between the number of iterations and the number of fitting test needed to solve a puzzle. However, both values decrease when the puzzle size increases and when the local informational context decreases. To address this point we have used a small context (10% of the border size) to solve 100 different puzzles with \(100 \times 100, 200 \times 200, 300 \times 300, 400 \times 400\) and \(500 \times 500\) pieces. Figure 8 shows the results of these tests. Both the number of iterations and the number of fitting tests increase with the puzzle size. However, the increment rate is larger for the SA. Therefore, the corresponding values for the distances \(d_{it}\) and \(d_{test}\) also increase with the puzzle size. For example, for puzzles of \(500 \times 500\) pieces \(d_{it} \approx 2,000\) iterations and \(d_{test} \approx 175 \cdot 10^9\) fitting tests. These values represent a 40% performance improvement by the SNN in both cases.
A large local context does not imply an optimal performance. When the context size is increased, the total number of fitting test needed to solve the puzzle is very similar to the one needed with a small context (right panels of Fig. 7). In the limit, for a context size close to the total number of neurons in the network, the local processing becomes equivalent to the global processing of the SA. In this case, the problem can be solved in only a few iterations, but this does not mean that the number of fitting test decreases. Based on all our measurements, we can conclude that the best performance of the SNN (in terms of both the number of fitting tests and iterations) is achieved for large puzzles using a relatively small context. This combination provides the optimal balance between the number of iterations and the number of comparisons to efficiently solve the problem.

V. DISCUSSION

In this paper we have introduced a self-organizing neural network paradigm that is able to discriminate information locally using a strategy for information processing inspired in recent findings in living neural systems. The network uses neural signatures to identify each unit, a transient memory in each cell to keep track of the information and its sources, and a multicoding mechanism for information propagation. This provides the ability to discriminate inputs in each neuron during the processing. To illustrate that the proposed paradigm can use these strategies to efficiently solve a problem, we have defined a general framework for multidimensional sorting problems and we have applied it to a classical task: the assembly of jigsaw puzzles. We have compared the results of our new approach with a classical stochastic method to solve the problem, and we have pointed out the situations where the new paradigm improves the performance.

We have analyzed the performance of the proposed algorithm in terms of the effort needed to solve the problem according to two different measurements (number of iterations and number of fitting tests). In both cases we have found a similar result. Local information discrimination has a computational cost that is evident in small puzzles. For large puzzles this computational cost is justified as our results show that local discrimination provides a better performance.

Due to the nature of the jigsaw puzzle problem, we have limited our analysis to the case in which each neuron receives and processes one input message per iteration. This restriction allowed us to compare the SNN performance with a classical approach in equivalent conditions. If we consider multiple messages per iteration in the SNN, neurons can process a larger amount of information in parallel. In this case, the local information context can be built in different ways. In a multiple message scenario, we have randomly chosen different fragments of the inputs of each cell to build the context. This strategy leads to solve the problem in fewer iterations (see top panel of Fig. 10). However, the number of fitting tests required...
panels a and b), the opposite situation occurs. However, for large puzzles (and specially for small values of the context size)

is larger, as expected (bottom panel of Fig. 10). On the other hand, we have only used the local informational context to store neuron data from cells whose piece matching is to be tested. Alternatively, a “negative context” can be used to save temporary information about cells whose matching has already been tested with a negative result and thus this information is considered not useful for the neuron. This negative context not only reduces the number of fitting tests, but also the number of iterations.

In the context of the jigsaw puzzle problem, the SNN defines a different search than the general puzzle-solver search schema. Local discrimination allows to group pieces in clusters dynamically with no a priori information. Each cluster contains pieces that have a high probability to match. Regarding the problem of reassembling real 3D broken objects, this is an desirable property, because the fitting among fragments usually is more difficult than among pieces of a commercially produced jigsaw puzzle [40]. Some of the traditional algorithms try to group similar pieces to reduce the number of fitting test [5], [20], [15], [43]. However these approaches require a significant preprocessing. On the other hand, the processing rules of the SNN for the jigsaw puzzle could include the use of classical similitude metrics such as the concavity and convexity of border pieces. Using them together with the local information discrimination, they can significantly reduce the number of fitting test needed for the SNN to find the puzzle solution.

We would like to emphasize that the proposed paradigm has a wider use beyond the context of jigsaw puzzles. There is a large flexibility to implement the core concepts of the SNN, thus these networks can be adapted to solve different problems that can benefit from the local information discrimination. Depending on the specific problem, one might need to provide a good performance in terms of number of iterations and cost measurements. The SNN can provide in many cases a good balance between performance and computational cost. A straightforward application of SNN is multidimensional sorting when the order in a particular dimension can be independent of the order in other dimensions, or when there is no global sorting criteria in any dimension. The local discrimination of the SNN can contribute to provide an efficient solution to these problems once the right balance between its cost and the performance is found (through the specification of the size of the local informational context). Areas of application for this kind of sorting that are likely to benefit from the SNN approach are scheduling, planning and optimization [2], [7].

Note that the SNN uses a self-organizing strategy that includes a nonsupervised learning as a function of the local discrimination. In addition, signature neural networks allow for a new set of learning rules that can include not only the modification of the connections, but also the parameters that affect the local discrimination. Subcellular plasticity is also a characteristic that has been recently studied in the nervous system of the jigsaw puzzle problem, the SNN improves and the effectiveness is similar for both approaches (white regions where $d_{eff}$ is near 0). For small puzzles the SA solves the problem before the SNN reaches a minimum effectiveness level. However, for large puzzles (and specially for small values of the context size $d_{eff} < 0$), the opposite situation occurs.

![Fig. 9. Evolution of $d_{eff}$ as a function of the puzzle size for three different contexts. Dark colors indicate the cases where the SA has better effectiveness ($d_{eff} < 0$). On the other hand, light colors indicate better effectiveness for the SNN ($d_{eff} > 0$). Solid/dashed lines denote the average number of iterations needed to solve a puzzle of a specific size with the SAS/SNN paradigm (calculated over 100 puzzles). In the initial iterations the effectiveness of the SA is always better. Then the SNN improves and the effectiveness is similar for both approaches (white regions where $d_{eff}$ is near 0). For small puzzles the SA solves the problem before the SNN reaches a minimum effectiveness level. However, for large puzzles (and specially for small values of the context size $d_{eff} < 0$), the opposite situation occurs.](image-url)

![Fig. 10. Comparison between the performance of the SNN in monosynaptic information propagation mode (only one input channel per iteration and neuron) and in multisynaptic propagation mode (each neuron receives four input messages in parallel). Top panel: Performance in terms of the mean number of iterations needed to solve the puzzle. Bottom panel: Performance in terms of the mean number of fitting tests. In all cases the size of the local informational context is equal to 10%. All measures are calculated by solving 100 different puzzles for each border size. The large number of iterations for puzzles of size $30 \times 30$ pieces is due to the low informational context for this size. Note that this effect is reduced in the multisynaptic mode.](image-url)
system [9].

In the introduction we have mentioned that uniformity of neurons has facilitated the mathematical formulation of many ANN paradigms. Local discrimination is somehow a problem to achieve a compact formalization for the SNN paradigm since this formalization depends on the specific problem that the network is trying to solve. This does not mean that some concepts that underlie the strategy of SNN paradigm cannot be used to extend classical ANN. For example, in particular applications, we can consider having different sets of transfer functions for each unit, and make the selection of the specific function depend on the state of a local informational context. This strategy can combine synaptic and intra-unit learning paradigms and lead to achieve multifunctionality in the network.

There is an increasing amount of new results on the strategies of information processing in living neural systems [29]. Beyond the specific results reported in this paper, the use of novel bio-inspired information processing strategies can contribute to a new generation of neural networks with enhanced capacity to perform a given task.

APPENDIX

MULTIDIMENSIONAL SORTING EXAMPLE

To illustrate the strategy of the SNNs to solve multidimensional sorting problems and to detail the evolution of the SNN implementation described in section III-B, let us consider the multidimensional sorting presented in Fig. 11. For simplicity, we will describe the bidimensional case first. In this case, the final goal is to sort horizontally and vertically with a SNN 12 elements in the order shown in panel a. The order criteria is given by the compatibility between colors displayed in this panel.

In this SNN example (Fig. 11b), the neuron signature is the number of each neuron in the network. The neuron data are the elements to sort illustrated by the different colors. For simplicity, we choose \( N_{context} = 3 \), \( P = 100\% \) and \( T = 10 \). There is no global order criteria, so two neurons are compatible if their corresponding blocks are adjacent in Fig. 11a. We will consider a multisynaptic information propagation mode with a maximum of two active output channels per iteration.

Taking this into account, we describe below the evolution of the SNN presented in Fig. 11b during the first two iterations. To follow this example, we recommend the reader to have in mind the definitions of section II-A and Fig. 12.

A. Initialization of the SNN

The initialization of the network consists of:

1. Neuron data are initialized by randomly assigning an element to each neuron.
2. The initial structure of the network is two-dimensional with each cell connected to its four nearest neighbors. The SNN has periodic boundary conditions, each neuron of a border is connected to the neuron of the opposite side.
3. For each neuron of the network \( \rightarrow n_i \):

B. Discrimination and processing rules

After the initialization, neurons build and send initial messages and the SNN starts to search for the solution (see Fig. 12). For this task, neurons follow the algorithm described in section III-B. This example uses the following local discrimination rules:

- During the process synaptic input phase, in the second discrimination stage (step 1.5 of the algorithm), neurons to be processed are selected from messages of \( in \). To simplify our description, here we will assume that active channels are sampled one neuron information at a time in a clockwise order (top channel first, right second, bottom third and left fourth). For example, if we consider neuron \( N01 \) in the left panel of Fig. 12, the neuron receives message \( N10 \rightarrow N05 \rightarrow N09 \) from \( N09 \) (top channel) and message \( N08 \rightarrow N12 \rightarrow N04 \) (left channel) from \( N04 \). Right and bottom channels are not active in this case. As the signatures of \( N09 \) and \( N04 \) are not recognized, both messages pass to the second discrimination stage. To select the neuron informations to process from the selected messages, \( N01 \) initially chooses \( N09 \) (first neuron information from the top channel), \( N04 \) (first neuron information from the left channel) and \( N05 \) (second neuron information from the top channel). \( N09 \) and \( N05 \) are discarded because they belong to the local informational context of \( N01 \) (remember that at the beginning of this iteration the context of \( N01 \) is the one built during the context initialization: \( N09, N02 \))
and N05, see Fig. 11a). Then, as $N_{context} = 3$ in this example, N01 chooses N12 (second neuron information from the left channel) and N10 (third neuron information from the top channel), to finally process N04, N12 and N10.  

- If a receptor starts recognizing a signature in a given iteration, it sends a message to this emitter. As we have a maximum number of active channels, these messages have a higher priority during the information propagation phase (step 2.1 of the algorithm). In the example, N01 starts recognizing the signature of N04 in iteration #1 (left panel of Fig. 12). Then, it sends a message to N04. The rest of output channels are activated randomly taking into account that the communication is bidirectional and the maximum value of active channels.

- During the information propagation phase, the emitter does not include information about the receptor in the output message. For example, let us consider neuron N01 whose context is N04, N12 and N10 at the end of iteration #1 (shown below the neuron in Fig. 12). N01 sends to neuron N04 the message $N10 - N12 - N01$ instead of $N12 - N04 - N01$.

C. Evolution of the SNN

1) Iteration #1: The left panel of Fig. 12 illustrates the evolution of the SNN presented in Fig. 11b during the first iteration of the algorithm. The first step during the process synaptic input phase consists in the selection of the messages to process. In iteration #1 there are no messages received from an emitter with a recognized signature, because no neuron recognizes any signature yet. Therefore, all messages pass to the second discrimination stage. Now, each neuron selects the set of neuron informations to process following the rules described in the text. The selected set is shown below each neuron, and this will be the local informational context for the next iteration.

After the initialization, the first messages are built and sent (not shown here). In our case, N01 sends messages to N09 and N04; N02 to N10 and N03; N03 to N02 and N04; N04 to N05; N05 to N06 and N08; N06 to N10 and N05; N07 to N11; N08 to N04 and N05; N09 to N01 and N12; N10 to N06 and N05; N11 to N02, N03 and N05; and N12 to N09. During the process synaptic input phase of iteration #1, all incoming messages will pass to the second discrimination stage. When a neuron starts recognizing a signature, the corresponding cell is shown green and filled. If the cells are also in their correct position, the connection between them are green and solid instead of red and dotted. Neurons filled and grey have a new connection established during the restore neighborhood phase. Arrows denote the output channels activated during the propagate information phase. Right panel: Evolution of the SNN in iteration #2. The differences with respect to the network in the left panel consist in the yellow filled neurons. These are cells moved to their correct position because a receptor receives a message from an emitter whose signature is recognized (and the emitter is not in its correct position).
mational context of N02, N02 has not a bottom neighbor and N04 has not a top neighbor: a new connection is established between these cells. Note that although N02 and N09 could be connected to complete their neighborhood (N02 is included in the local context of N09) they are not connected because they are already neighbors. Note that the neighborhood restoration takes place after the local informational context is updated.

2) Iteration #2: The second iteration starts from the situation shown in the left panel of Fig. 12. The mechanism used to process the incoming messages in the receptor is analogous to the mechanism described for iteration #1 with only one difference. Now, N01 receives a message from N04, an emitter with a recognized signature and not its correct position. Then, N01 only processes this message. Before processing the message, as N04 is not in its correct position, a reconfiguration of the network takes place to move N04 on top N01 (right panel of Fig. 12). At this point N04 does not recognize signature S01 yet. Later, when N04 processes its input message it will start recognizing S01.

During the process synaptic input phase, N04 processes, in this order, the neuron information of cells N01, N03 and N12. The processing of neuron information of N01 implies that N04 starts recognizing signature N01. As N01 belongs to the neighborhood of N04 a reconfiguration is not needed. The same occurs when the neuron information of N03 is processed. Finally, the processing of neuron information of N12 also implies that N04 starts recognizing signature N12. But now, N12 does not belong to the neighborhood of N04. When a new synapse is established to set N12 in its correct position, the connection between N03 and N04 must be broken. As a consequence, N04 stops recognizing signature N03 and, N03 stops recognizing N04.

In this simple example, in just two iterations most of the SNN neurons have reached the local solution (c.f. Fig. 11c); and only two more iterations are needed to reach the global solution to the problem (not shown here).

Note that the corresponding problem in three dimensions (illustrated in panel d in Fig. 11) or more only requires to repeat step 1.6 of the algorithm described in section III-B.

ACKNOWLEDGMENT
This work was supported by MICINN BFU2009-08473, TIN2007-65989 and CAM S-SEM-0255-2006.

REFERENCES


Roberto Latorre received his degree in Computer Engineering in 2000 and the PhD in Computer Science and Telecommunications in 2008 from Universidad Autónoma de Madrid. He is member of the Grupo de Neurocomputación Biológica (GNB) at the Escuela Politécnica Superior since 2001. His research interests span different topics in neuroscience and neurocomputing, from the generation of motor patterns and information coding to pattern recognition and artificial neural networks. Since 2002 he is professor asociado at the Escuela Politécnica Superior, Universidad Autónoma de Madrid.

Francisco de Borja Rodríguez received his degree in Applied Physics in 1992 and the PhD in Computer Science in 1999 from Universidad Autónoma de Madrid. He then worked at the Nijmegen University in Holland and the Institute for Nonlinear Science, University of California, San Diego. Since 2002 he is profesor titular at the Escuela Politécnica Superior, Universidad Autónoma de Madrid.

Pablo Varona received his degree in Theoretical Physics in 1992 and the PhD in Computer Science in 1997 from Universidad Autónoma de Madrid. He was a postdoc and later an assistant research scientist at the Institute for Nonlinear Science, University of California, San Diego. Since 2002 he is profesor titular at the Escuela Politécnica Superior, Universidad Autónoma de Madrid.