Exploiting Natural Asynchrony and Local Knowledge within Systemic Computation to Enable Generic Neural Structures

Erwan Le Martelot[1,3], Peter J. Bentley[2], and R. Beau Lotto[3]

[1] Engineering Department, University College London, London, UK,
e.le_martelot@ucl.ac.uk,
[2] Computer Science Department, University College London, London, UK,
[3] Institute of Ophthalmology, University College London, London, UK

Abstract. Bio-inspired processes are involved more and more in today's technologies, yet their modelling and implementation tend to be taken away from their original concept because of the limitations of the classical computation paradigm. To address this, systemic computation (SC), a model of interacting systems with natural characteristics, followed by a modelling platform with a bio-inspired system implementation were introduced. In this paper, we investigate the impact of local knowledge and asynchronous computation: significant natural properties of biological neural networks (NN) and naturally handled by SC. We present here a bio-inspired model of artificial NN, focussing on agent interactions, and show that exploiting these built-in properties, which come for free, enables neural structure flexibility without reducing performance.

1 Introduction

An ant colony is driven by individual agents acting individually, asynchronously and randomly, yet can accomplish complex and precise tasks. It is an example of a non-analytical and natural process performing local computation that leads to a precise and complex global result. Similar to biological systems, artificial bio-inspired approaches suggest some inner natural characteristics. We can state that natural processes are stochastic, asynchronous, parallel, homeostatic, continuous, robust, fault tolerant, autonomous, open-ended, distributed, approximate, embodied, complex, have circular causality and compute locally [1]. Such characteristics are not natively present in current conventional paradigms and models of natural processes that run on conventional computers have to include a simulation of some of these features. This often leads to slower and less straightforward implementations compared to analytical or linear algorithms for which computers are well suited.

Just as the development of Prolog enabled elegant and precise implementations of logical expressions, so the development of a paradigm where processes could be defined in a manner that resembles their true structures would improve our ability to implement bio-inspired processes. To address this, [1] introduced

Systemic Computation (SC), a new model of computation and corresponding computer architecture based on a systemics world-view and supplemented by the incorporation of natural characteristics. This work was followed by the introduction of a complete platform for this paradigm [2].

In this paper, we focus on two natural properties of SC: local knowledge and asynchronous computation, applying them to a common bio-inspired paradigm: artificial neural networks (ANN). Local knowledge and asynchrony do not suit conventional computer architectures and classical ANN models often employ global algorithms, constraining the network structure and making them less biologically plausible. Real biological NN imply a more flexible model without the structural limitations imposed by conventional approaches. We thus suggest an ANN implementation using SC. The use of SC requires the use of local knowledge and asynchronous computation. We show that such a model enables the implementation of the same networks as those implemented using conventional global and synchronous approaches, but the SC implementation does not constrain the network structure. We then compare our approach to more classical ones.

2 Motivation and Background

There have been alternative views of computation since its conception: cellular automata have proven themselves to be a valuable approach to emergent, distributed computation; generalisations such as constrained generating procedures and collision-based computing provide new ways to design and analyse emergent computational phenomena; bio-inspired grammars and algorithms introduced notions of homeostasis, fault-tolerance and parallel stochastic learning; bio-inspired paradigms showed good modelling potential for biological systems. New architectures are also popular, like distributed computing (or multiprocessing), computer clustering, grid computing, ubiquitous computing and speckled computing.

To unify notions of biological computation and electronic computation, [1] introduced SC as a suggestion of necessary features for a computer architecture compatible with current processors, yet designed to provide native support for common characteristics of biological processes. Table 1 lists major characteristics that can be found in some computation paradigms. It shows the inner properties (i.e. those that need not be simulated) of natural computation (e.g., ants, neurons, DNA), systemic computation, conventional programming languages (procedural, object-oriented, functional and logical), and bio-inspired paradigms like cellular automata (CA) or membrane computing (P-systems). Table 1 illustrates the proximity between SC and natural computation compared to other common computational paradigms. Previous work [2] showed how simply and naturally a bio-inspired process such as a GA could be created and given more autonomy being made self-adaptive by just adding a single component. Defining systems and interactions only and following the SC rules thus naturally lead to an easily evolvable, stochastic, approximate, continuous and complex process. To provide evidence that processes can also benefit from built-in asynchronous computation and local knowledge, we focus in this paper on artificial neural networks (ANN).

	Nature	S.C.	C. Lang.	C.A.	P-systems
Stochastic (Deterministic)	Y (N)	Y (N)	N (Y)	N (Y)	Y (N)
Asynchronous(Synchronous)	Y (N)	Y (N)	N (Y)	N (Y)	N (Y)
Parallel (Serial)	Y (N)	Y (N)	N (Y)	Y (N)	Y (N)
Continuous (Batch)	Y (N)	Y (N)	Y/N(Y/N)	Y (N)	Y (N)
Distributed (Centralised)	Y (N)	Y (N)	N (Y)	Y (N)	Y (N)
Approximate (Precise)	Y (N)	Y (N)	N (Y)	N (Y)	Y (N)
Embodied (Isolated)	Y (N)	Y (N)	N (Y)	N (Y)	N (Y)
Circular (Linear) causality	Y (N)	Y (N)	N (Y)	N (Y)	Y/N (Y/N)
Local (Global) knowledge	Y (N)	Y (N)	Y/N(Y/N)	Y (N)	Y (N)

Table 1. Features of various computational paradigms. 'Y' indicates the characteristic is built-in and needs no extra implementation; 'N' indicates that extra implementation is needed to simulate the characteristic; 'Y/N' means the method is capable of supporting models that may have or not have the property without extra implementation.

ANN are suitable to highlight the aforementioned properties as:
- neurons are organised to create a whole (the network) that solves problems,
- neurons are computing locally, yet the result is global,
- neurons are independent (in timing and internal knowledge).

Classical backpropagation (BP) [3] constrains the network to be layered and feed-forward; therefore no change in the neurons' organisation breaking this requirement can be made. Recurrent BP was introduced to overcome one of these constraints and cope with backward connections [3]. Other more biologically plausible techniques, like contrastive Hebbian learning for deterministic networks [5][6], generalised recirculation [8], or spiking neurons networks [7] were introduced and showed successful results. Still, these approaches all define global algorithms, coping with various specific network structures, giving neurons more and more realistic computational abilities, but do not give the neuron entity the ability to be autonomous (i.e. inner data processing) in whatever situation (i.e. disregarding the position in the structure). Such natural flexibility is, from our modelling point of view, what is desirable and missing in approaches using conventional computation. The reason for using SC at all is to move beyond simply attempting to mimic the functional behaviour of natural systems through global algorithmic approximations, and instead (as much as is feasible) duplicate the functional behaviour through mirroring the underlying systems, organisations and local interactions. SC is thus intended to be a truer representation and thus an improved model of natural systems implemented following its paradigm, compared to other approaches.

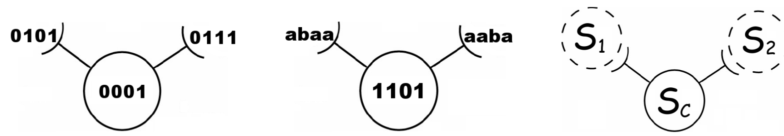
3 Overview of Systemic Computation

SC [1] is a new model of computation and corresponding computer architecture based on a systemics world-view and supplemented by the incorporation of natural characteristics (previously listed). This approach stresses the importance of structure and interaction, supplementing traditional reductionist analysis with the recognition that circular causality, embodiment in environments and emergence of hierarchical organisations all play vital roles in natural systems. Systemic computation makes the following assertions:
- Everything is a system.
- Systems can be transformed but never destroyed.
- Systems may comprise or share other nested systems.
- Systems interact, and interaction between systems may cause transformation of those systems, where the nature of that transformation is determined by a contextual system.
- All systems can potentially act as context and affect the interactions of other systems, and all systems can potentially interact in some context.
- The transformation of systems is constrained by the scope of systems, and systems may have partial membership within the scope of a system.
- Computation is transformation.

In systemic computation, everything is a system, and computations arise from interactions between systems. Two systems can interact in the context of a third system. All systems can potentially act as contexts to determine the effect of interacting systems. A system is divided into three parts: two schemata and one kernel. These three parts can be used to hold anything (data, typing, etc.) in binary as shown in Figure 1(a). The kernel defines the result of two systems interacting in its context (and may also optionally hold data if it is interacting with another system). The two schemata define which subject systems may interact in this context as shown in Figures 1(b) and 1(c). A system can also contain or be contained by other systems. This enables the notion of scope. Interactions can only occur between systems within the same scope. An SC program therefore comprises systems that are instantiated and positioned within a hierarchy (some inside each other). It thus defines an initial state from which the systems can then randomly interact, transforming each other through those interactions and following an emergent process rather than a deterministic algorithm. For full details see [1] and [2].

4 ANN Model

Modelling a neural network keeping all its natural characteristics should involve the same entities that form a real one: neurons, their inner mechanism and their communication mechanism. These mechanisms could be modelled a priori at several levels. One model could represent the interaction of neurons using synapses to make the link between axon and dendrites. Another one could involve pre-synapse, post-synapse, protein exchange, protein transfer, etc. We chose to study



(a) Data system (b) Context system (c) Interaction

Fig. 1. 1(a): A system used primarily for data storage. The kernel (in the circle) and the two schemata (at the end of the two arms) hold data. 1(b): A system acting as a context. Its kernel defines the result of the interaction while its schemata define allowable interacting systems. 1(c): An interacting context. The contextual system S_c matches two appropriate systems S_1 and S_2 with its schemata and specifies the transformation resulting from their interaction as defined in its kernel.

and create our model at the neuron level of abstraction and not explicitly represent protein interactions. A neuron receives inputs from its dendrites that are processed in the soma; the resulting signal is then sent through the axon [4]. Axon signals are weighted and transmitted to further neurons through synapses which communicate with their dendrites. The signal will thus be a value transmitted across the network rather than many molecular and electrical entities.

4.1 Systemic Analysis

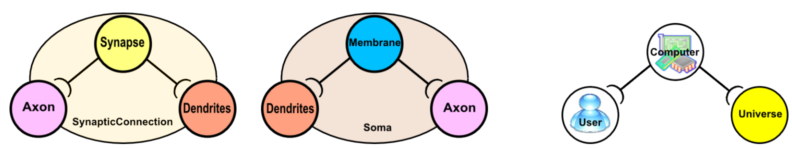
Using SC implies a systemic analysis of the problem to define the interacting systems and how they interact [1]. The synapse which transfers signals from axon to dendrites can be chosen as a context of interaction between neurons. However, neurons interaction do not provide information regarding the signal flow direction. This flow is by definition directional from axons to dendrites. Therefore the model should have the more precise notions of axons and dendrites to precise the signal direction. Dendrites can be modelled as one system representing the dendritic tree rather than one system per dendrite which would add unnecessary complexity to the model. A synapse connecting an axon with a dendrites system, each systems triplet belongs to the scope of a connection (Figure 2(a)).

Two types of synapses could be considered here: excitatory and inhibitory synapses [4]; not to mention that synapses can be electrical or chemical [4], which we do not explicitly model here. For modelling simplicity and not to introduce inconsistencies we chose to allow both excitatory and inhibitory excitations within one synapse. This is modelled by a weight taken within [-1; 1]. A positive weight simulates an excitatory synapse and a negative weight an inhibitory one.

To model the signal processing between dendrites and axon inside a neuron, we can consider the ionic transmissions in the membrane and the membrane as a whole and define the membrane as context of interaction between dendrites and axon, as shown in Figure 2(b). A membrane also owns a threshold of signal activation, real value also taken within [-1; 1].

To keep neuronal integrity, scopes are used to group what is part of a neuron, of the outside or of both. All the inherent neuron interactions happen within its soma. A neuron is therefore represented as dendrites, a soma, a membrane and an axon. However, dendrites and axons also belong to the outside (they are exposed to the outside of the soma) as their role is to receive and transmit signals from or to other neurons. Therefore, neurons can be modelled as shown in Figure 2(b).

Neurons belong to a NN, therefore it is sensible for integrity to encompass them in a “network” system itself contained in the systemic “universe”. The universe is here a system which encloses everything within the program. It is also used as the interface between the program and the user (Figure 2(c)). However, the network inputs and outputs as well as the data transfer between them and the universe are still to be defined. A real brain receives axons from neurons located outside, like visual inputs, and sends signals also outside, like to muscles. Thus, axons can naturally also play the role of network inputs and outputs. Then “InputTransfer” (IT) and “OutputTransfer” (OT) context systems transfer data between the universe and the input and output axons. Figure 3 shows a single neuron systemic neural network.



(a) (b) (c)

Fig. 2. 2(a): Axon-dendrites interaction in the context of a synapse. 2(b): Systemic model of a neuron showing the dendrites-axon interaction in the context of a membrane, and within a soma. 2(c): User-program interaction in the context of the computer.

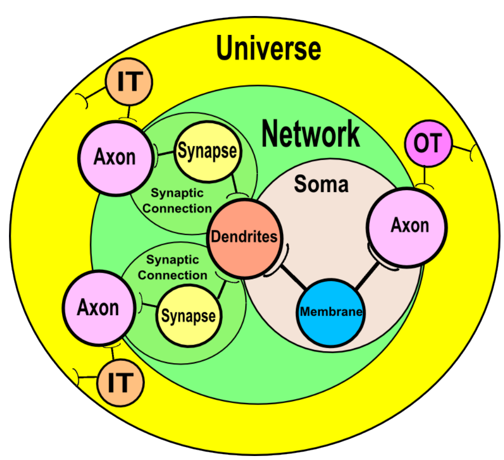


Fig. 3. Systemic NN with 2 inputs and 1 neuron.

So far, this model can organise interactions disregarding the physical location of the neurons. Nonetheless, the notion of neuron neighbourhood can be easily handled using scopes. An "area" system can encompass neurons and neurons can belong to several areas. This partition and sharing of neurons would thus create neighbourhoods in the network. Note that the physical neighbourhood is defined by relationships between systems rather than by physical coordinates. Figure 4 shows a more complex network with areas and recurrent connections.

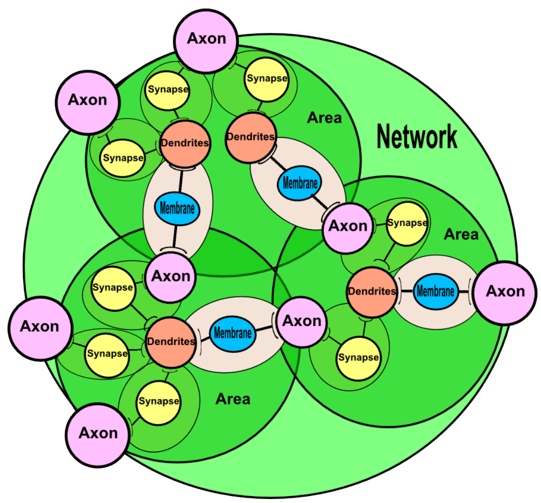


Fig. 4. Network with four inputs, one output and three areas sharing neurons. Each area defines a neighbourhood for the neurons inside.

This network partitioning into areas using scopes also offers potential interest for future work. Some more interaction possibilities could then be added, injecting new context systems in specific areas, thus giving one a different potential and behaviour from another. In addition, from a biological modelling point of view, partitioning the network into areas is of relevance [4].

4.2 Rules

The organisation of neurons is based on observations taken from biological studies [4]. However, knowing the organisation does not explain the inner behaviour of the entities involved. Unfortunately, this is not well understood yet how everything happens at this stage. We are thus forced to use methods that may or may not be biologically plausible, and use an adaptation for asynchronous and local computation of the gradient back propagation (BP) method [3] for learning. BP is often described as a global algorithm relying on some precise network structure [3]. The aim of our adaptation is to keep the principle of this method but adapt it to be a local-rule based principle.

BP relies on the concept of layers to group independent neurons together, which provides an easy control of the flow of information from layer to layer and therefore suits a global and serial algorithm. In the SC paradigm, we can use the very same principle without any structure hypothesis by defining the information flow process locally. Equations 1 to 3 give the BP rules with a momentum factor:

(1) $x_i = g(h_i) = g(\sum_k w_{ik}.x_k)$

(2) $\Delta w_{ij}(t) = \lambda.e_i.x_j + \alpha.\Delta.w_{ij}(t-1)$

(3) $e_i = g'(h_i).\sum_j w_{ji}.e_j = \sum_j (g'(h_i).w_{ji}.e_j)$

with x_i, h_i and e_i the signal output, the weighted input sum and the backpropagated error value of neuron i, w_{ij} the weight from neuron j to neuron i, λ the learning rate, α the momentum term, g the transfer function and g' its gradient.

The mathematical principles of the rules can be kept. However their implementation needs to be local. Equation (3) shows that the error can be written as a sum. It can therefore be performed by a sequence of independent computations as long as their common term $g'(h_i)$ remains constant during computations. Figure 5 shows a flowchart of the error backpropagation and delta weight update.

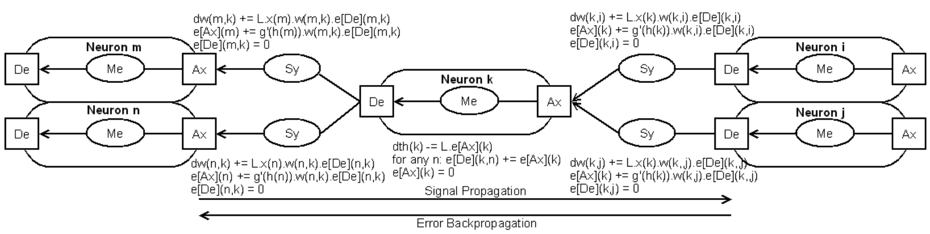
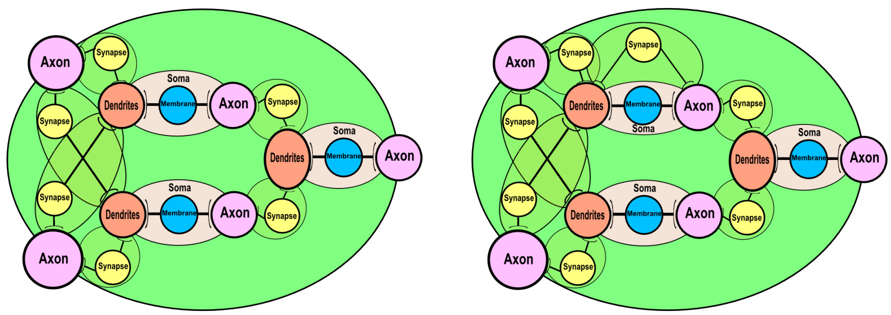


Fig. 5. Neuron k receives error values from ahead neurons i and j and backpropagates error values to neurons m and n. Error is transmitted between dendrites (De) and axons (Ax) by the context systems membrane (Me) and synapse (Sy). A synapse updates its delta weight (dw), a membrane its delta threshold (dth), L is the learning rate and e is the error. The momentum term is not shown for simplicity and readability.

Each neuron keeps a current error value where further neurons can add their backpropagated error contribution. To avoid reusing twice any information, a neuron resets to zero its error value as soon as it is backpropagated. To assert the constancy of the terms $g'(h_i)$, the weights are updated only at the end of the computation (like in a classical BP) when a new sample is presented.

Neurons and synapses are therefore autonomous and responsible for their own local data. These local rules imply that no global algorithm is defined and therefore no constraint is imposed on the network structure. This model can be used to design feed-forward networks as well as recurrent NN as shown in 6(a) and 6(b). The biological plausibility comes in this work from the autonomy and organisation of the systems, caused by the natural characteristics of asynchrony and local knowledge built into SC, leading to an emerging global behaviour, like the global learning. Also, the model could use any other kind of learning within the neuron and synapse systems, still keeping the very same organisation.

Note that a stabilised network could easily have its weak synapses trimmed by the injection of a new context system, programmed for instance to kill settled redundant synapses. This illustrates how the model could be improved by easy addition of new systems rather than requiring modifications of code at its core.



(a) (b)

Fig. 6. 6(a): A feed forward network. 6(b): Same network with a recursive synapse. The program is initially the same but then topped up with one more synapse. (Synaptic Connection systems are made discreet for readability but the synapses schemata clearly indicate the interacting systems.)

5 Experiments

5.1 Experiment 1 - X-Or

The X-Or problem is a common example of non-linearly separable problems, problems simple perceptrons cannot solve. Usually, BP feed-forward multi-layer perceptrons are used [3] and a common solution involves 2 hidden neurons and 1 output neuron (Figure 7(a)) [9]. However, the X-Or problem becomes a linearly separable problem when adding a third input, doing for instance the AND of the 2 others [9]. A 2 neurons network is thus enough to solve the X-Or problem [9] (Figure 7(b)). These structures can be simply created with our SC model by just assembling neurons together (i.e., adding the appropriate dendrite, soma and axon systems into the environment). In the experiments, both network structures were created and their performance was evaluated with the new, local BP rules.

We compare these results to the network 7(a) using the Matlab ANN toolbox. We instantiate a network using a hyperbolic tangent sigmoid transfer function ($sig(x) = \frac{2}{1+e^{-2.x}} - 1$) on the hidden layers and a linear transfer function on the output node. The rest of the setup is the toolbox's default (Matlab 7.1, NN Toolbox 4.0.6). There is no Matlab implementation of the network 7(b) as

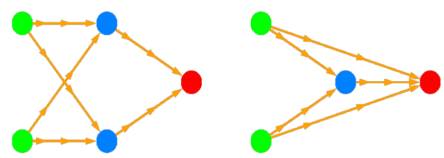


Fig. 7. 7(a) and 7(b): 2 networks for the X-Or problem.

classical BP only deals with layered networks. The SC networks use the same transfer function as in the Matlab network for each node including the output one. However, it sets the error on the output node using the gradient of the identity function (i.e. $g'(h) = 1$). The gradient of a sigmoid function on the last node provides poor error estimation as a sum around the worst value (i.e. 1 instead of -1) is little penalised ($g'(h) \approx 0$) compared to a neutral value ($g'(h) >> 0$). The learning rate is set to 0.5 with no momentum. The networks are allowed 100 epochs for learning. Each test is run 25 times. Table 2 summarises the results. The binary input values are -1 for false and 1 for true. The results show that both SC networks outperform the Matlab implementation by a wide margin.

Matlab 7(a)	S.C. 7(a)	S.C. 7(b)
47%	84%	100%

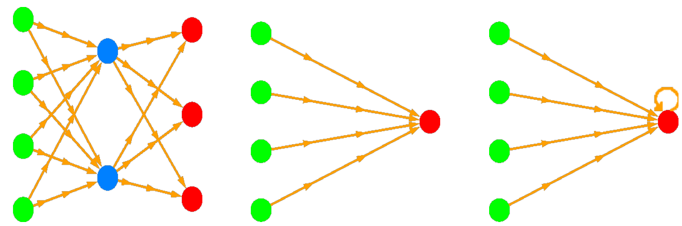
Table 2. X-Or experiments results giving for each network implementation (7(a) and 7(b)) the percentage of success (perfect truth table) over 25 runs in solving the problem.

5.2 Experiment 2 - Iris data

The second experiment uses the well-known Iris plant data set [10]. The data set contains three classes of fifty instances each, where each class refers to a type of iris plant (Setosa, Versicolour and Virginica). One class is linearly separable from the other two; the latter are not linearly separable from each other.

The network we use in both the Matlab and SC implementations has 4 inputs, 2 hidden neurons and 3 output neurons, as shown in Figure 8(a). The learning and momentum rates are respectively taken at 0.1 and 0.75. The maximum number of epochs was set to 50.

Table 3 shows the classification results over 10 runs taking half of the samples (75) for learning and the other half (75) for testing. The simulation in Matlab used the same setup and default settings. The Matlab implementation, learning in batch mode, required many more epochs to classify correctly; we set the limit at 1000. Then, the final error level is fairly similar. The results clearly show that our SC model can perform at least as well as conventional networks implementation in such classification task.



(a) (b) (c)

Fig. 8. 8(a):Feed forward 4-2-3 network for Iris plant data classification. 8(b): Perceptron, 8(c): Amplified perceptron.

	Setosa		Versicolour		Virginica		Total		
	L	T	L	T	L	T	L	T	L&T
Matlab	100	100	96.4	94	98	95.6	98.13	96.53	97.33
S.C.	100	100	93.2	94	99.2	97.6	97.47	97.2	97.33

Table 3. Averaged percentage of success over 10 runs of the 3 Iris data sets and total percentage of success (L and T respectively stand for the learning and the testing sets).

5.3 Experiment 3 - Recursive ANNs

Another advantage of the SC model is that it can also handle recursive connections. One example of such a network is a simple signal amplifier (i.e. where the signal of a neuron is reinforced by itself). In this third experiment, we made the two networks shown in Figures 8(b) and 8(c) learn a small amount of two sets to classify (Setosa and Versicolour, linearly separable). Each set consists of 50 samples and we give 5 of them for each set to the networks to learn within 3 epochs. We then test the networks with the rest of the sets samples. The learning rate is 0.25 and no momentum factor is used. Table 4 provides the results.

		Perceptron		Amplified Perceptron	
		Average	Std. Dev.	Average	Std. Dev.
L	Setosa	0.8982	0.0148	0.9370	0.0119
	Versicolour	-0.8076	0.1314	-0.8645	0.0838
T	Setosa	0.8769	0.0435	0.9264	0.0260
	Versicolour	-0.6092	0.2578	-0.6982	0.2258

Table 4. Average classification value for the Setosa and Versicolour (repectively taught at 1 and -1) sets over 10 runs with a simple perceptron and an amplified perceptron. "Average" gives the network response: the closer to the taught value, the stronger the belief in the classification. "Std.Dev." gives the standard deviation over the samples of each set. L and T respectively refer to the learning and testing sets.

Learning has been performed on very few epochs and samples; the network should thus be insensitive to noise. The signal reinforcement allows a strong response in a short learning time without requiring a steeper sigmoid function.

6 Conclusion

In this paper we continued our exploration of systemic computation, a novel paradigm designed to improve our ability to model and implement biological processes. We showed how its intrinsic (non-simulated) properties of local knowledge and asynchrony naturally provide more flexibility for artificial neural network structures. The example implementation contrasts significantly with classical approaches where data and algorithm are interdependent separate parts making network implementations more rigid and less biologically plausible. Our implementation gave full autonomy to neurons, and is compatible with any neuron model (first, second, third generation [7]). It thus highlights the potential of SC for the modelling of such natural processes. Future work will explore other features of SC such as fault tolerance and self-repair.

References

1. Bentley, P. J.: Systemic computation: A Model of Interacting Systems with Natural Characteristics. Int.J. Parallel, Emergent and Distributed Systems **22** (2007) 103–121
2. Le Martelot, E., Bentley, P. J., and Lotto, R. B.: A Systemic Computation Platform for the Modelling and Analysis of Processes with Natural Characteristics. In Proc of Genetic and Evolutionary Computation Conference (2007) 2809–2816
3. Tang, H., Tan, K. C. and Yi, Z.: Neural Networks: Computational Models and Applications. Springer (2007)
4. Kandel, E. R., Schwartz, J. H. and Jessel, T. M.: Principles of Neural Science (third edition), Chap 1,3, Elsevier (1991)
5. Peterson, C. and Anderson. J. R.: A Mean Field Theory Learning Algorithm for Neural Networks. Complex Systems **1** (1987) 995–1019
6. Hinton, G. E.: Deterministic Boltzmann Learning Performs Steepest Descent in Weight-space. Neural computation **1** (1990) 143–150
7. Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural Networks **10** (1997) 1659-
8. O'Reilley, R. C.: Biologically Plausible Error-driven Learning using Local Activation Differences: The Generalized Recirculation Algorithm. Neural computation **8** (1996) 895–938
9. Yanling, Z., Bimin, D. and Zhanrong, W.: Analysis and Study of Perceptron to Solve XOR Problem. Proc. of the 2nd Int. Workshop on Autonomous Decentralized System (2002)
10. Iris Plants Database, Creator: R.A. Fisher, Donor: Michael Marshall, (1988), UCI Machine Learning Repository [http://www.ics.uci.edu/ mlearn/MLRepository.html]