Archeologia e Calcolatori
Supplemento 6, 2014, 53-84

THE GENERAL PHILOSOPHY OF ARTIFICIAL ADAPTIVE SYSTEMS (AAS)

1. Artificial Adaptive Systems

Artificial Adaptive Systems (AAS) form part of the vast world of Natural Computation (NC) which is itself a subset of the Artificial Sciences. Artificial Sciences are those sciences for which an understanding of Natural and/or Cultural processes is achieved by the recreation of those processes through automatic models. We shall use an analogy to explain the difference between Artificial Science and natural language; the computer is to the Artificial Sciences as writing is to natural language. That is, the AS consists of a formal algebra used for the generation of artificial models which are composed of structures and processes, and natural languages are composed of semantics, syntax and pragmatics for the generation of texts.

Through each of these very different systems a level of independence is created; in natural languages the utterances of sounds are fully dependent on the time in which the utterances are made, but by representing those utterances with writing they become independent from time, for written documents (in the form of books, manuscripts, typewritten pages, computer generated Output in the form of both digital and hardcopy, etc.) exist outside the dimension of time. They exist in the spatial dimension. Similarly, the computer achieves independence from the physical system through the creation of a model. Such models are automations of the original system and permit one to study the natural/physical system at any time, even if the original system no longer exists.

An example of such a system is the active eruption of a volcano or the tremors of an earthquake. Through extensive measurements of variables a model can be constructed that permits researchers to recreate the original volcanic activity or earthquake in a completely controlled environment by which variables of choice can be controlled. By using writing as an extension of a natural language permits the creation of cultural objects that, before onset of writing, were unthinkable. Such cultural objects are stories, legal texts, manuals, historical records, etc. In a similar manner the AS can create models of complexity that, before the construction of computers, were unthinkable. Natural languages and Artificial Sciences, in the absence of writing and the computer, are therefore limited. But a written document not based on a natural language, or an automatic model not generated by formal algebra, are little more than a set of scribbles (Fig. 1).

In the Artificial Sciences, the understanding of any natural and/or cultural process occurs in a way that is proportional to the capacity of the

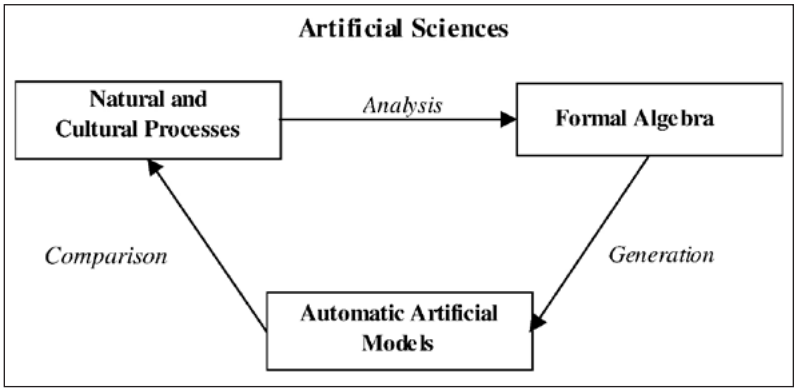

Fig. 1 – The diagram shows how the analysis of Natural and/or Cultural Processes, that need to be understood, starts from a theory which, adequately formalized (Formal Algebra), is able to generate Automatic Artificial Models of those Natural and/or Cultural Processes. Lastly, the generated Automatic Artificial Models must be compared with the Natural and/or Cultural Processes of which they profess to be the model and the explanation.

automatic artificial model to recreate that process. The more positive the outcome of a comparison between an original process and the generated model, the more likely it is that the artificial model has correctly explained the functioning rules of the original process. However, this comparison cannot be made simple-mindedly. Sophisticated analytical tools are needed to make a reliable and correct comparison between an original process and an artificial model. Most of the analytical tools useful for this comparison consist of comparing the dynamics of the original process and the dynamics of the artificial model when the respective conditions in the surroundings are varied. In sum, it could be argued that:

1) on varying the conditions in the surroundings, yields a greater variety of response dynamics obtained both in the original process and in the resulting artificial model; and
2) the more these dynamics between the original process and the resulting artificial model are homologous, we can therefore conclude
3) the more probable it is that the artificial model is a good explanation of the original process.

In Fig. 2, we propose a taxonomic tree for characterization of the disciplines that, through NC and Classic Computation, make up the Artificial Sciences system. NC refers to that part of the Artificial Sciences responsible for the construction of automatic models of Natural and/or Cultural Processes

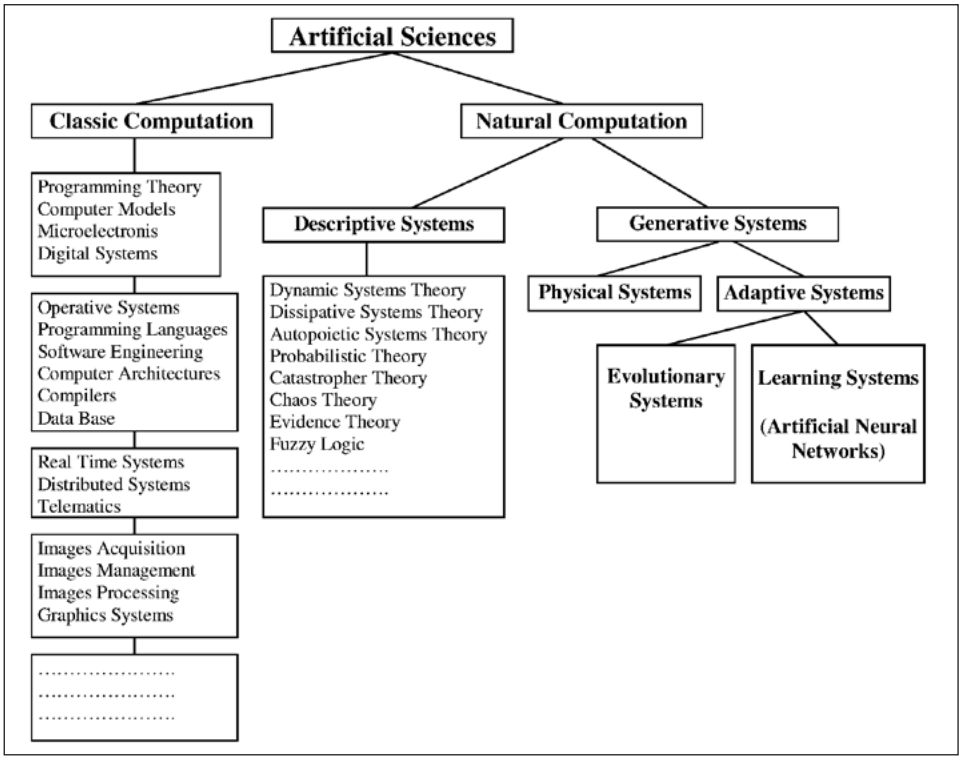

Fig. 2 – Taxonomic tree of the disciplines that make up the Artificial Sciences system.

through the local interaction of non-isomorphic micro processes. In NC, it is therefore assumed that:

1) every process is, more or less, contingent on the result of more basic processes that tend to self-organize in time and space;
2) none of the micro processes are themselves informative concerning the function that they will assume with respect to others, nor the global process of which it will be part.

This computational philosophy, very economic for the creation of simple models, can be used effectively to create any type of process or model that is inspired by complex processes. NC in fact deals with the construction of artificial models that do not simulate the complexity of Natural and/or Cultural Processes through rules, but rather, through commitments that, depending on the space and time through which the process takes form, autonomously

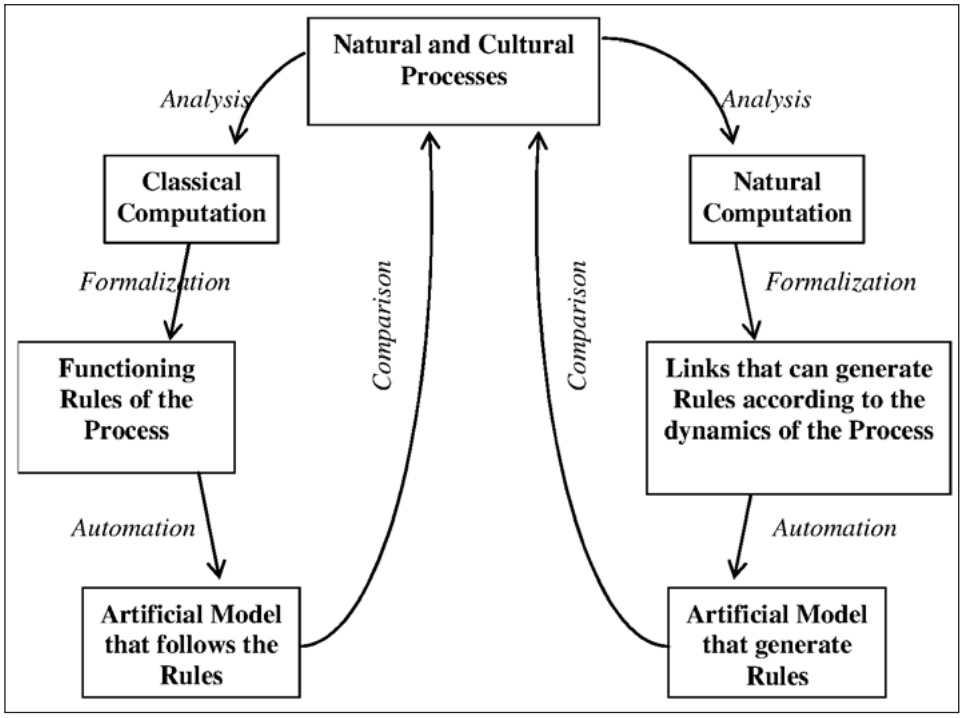

Fig. 3 – The diagram shows in more detail the formalization, automation and comparison between Natural and/or Cultural Processes and Automatic Artificial Models seen from two points of view (Classical Computation and NC). Each point of view can be seen as a cycle that can repeat itself several times. This allows one to deduce that the human scientific process characterizing both the cycles resembles more the NC than the Classical Computation one.

create a set of contingent and approximate rules. NC does not try to recreate Natural and/or Cultural Processes by analyzing the rules which make them function, and thus formalizing them into an artificial model.

On the contrary, NC tries to recreate Natural and/or Cultural Processes by constructing artificial models able to create local rules dynamically and therefore capable of change in accordance with the process itself. The links that enable NC models to generate rules dynamically are similar to the Kantian transcendental rules: these are rules that establish the conditions of possibility of other rules. In NC, dynamics such as learning to learn are implicit in the artificial models themselves, whilst in Classical Computation additional rules are required (Fig. 3). NC can be decomposed into the following:

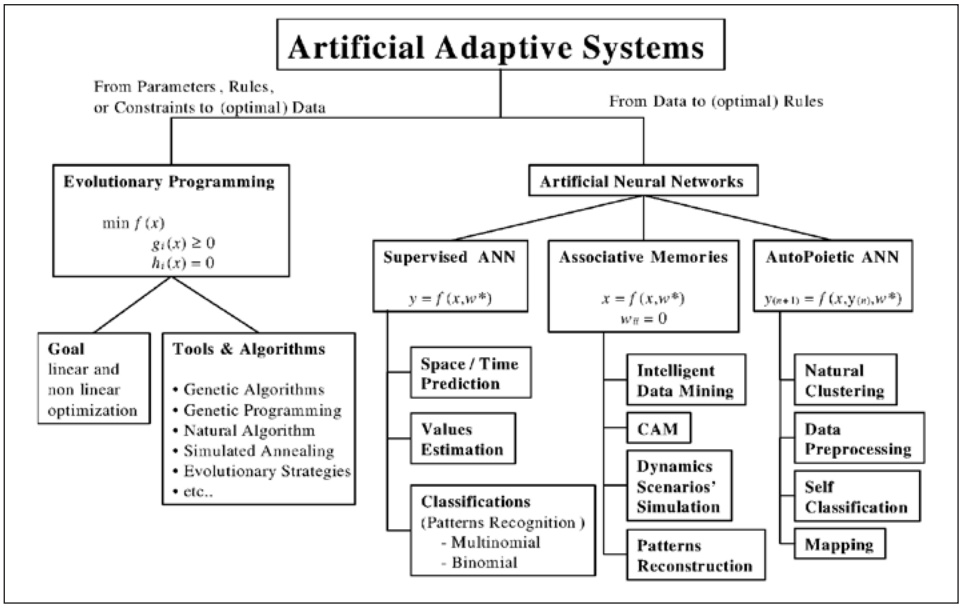

Fig. 4 – Artificial Adaptive Systems: general diagram.

– Descriptive Systems (DS): are derived from disciplines that have developed, whether or not intentionally, a formal algebra that has proved particularly effective in drawing up appropriate functioning links of artificial models generated within NC (for example: the Theory of the Dynamic Systems, the Theory of Autopoietic Systems, Fuzzy Logic, etc.).
– Generative Systems (GS): theories of NC that have explicitly provided a formal algebra aimed at generating artificial models of Natural and/or Cultural Processes through links that create dynamic rules in space and in time. In turn, Generative Systems can be broken down into:
– Physical Systems (PS): a grouping of those theories of NC whose generative algebra creates artificial models comparable to physical and/or cultural processes, only when the artificial model reaches given evolutionary stages (limit cycles type). Whilst not necessarily the route through which the links generate the model, it is itself a model of the original process. In brief, in these systems in which the generation time of the model is not necessarily an artificial model of the evolution of the process time (for example: Fractal Geometry, etc.).
– Artificial Adaptive Systems (AAS): theories of NC whose generative algebra creates artificial models of Natural and/or Cultural Processes, whose birth

process is itself an artificial model comparable to the birth of the original process. They are therefore theories assuming the emergence time in the model as a formal model of the process time itself.

In short: for these theories, each phase of artificial generation is a model comparable to a Natural and/or Cultural process. Artificial Adaptive Systems in turn comprise (Fig. 4):

– Learning Systems (Artificial Neural Networks – ANNs): these are algorithms for processing information that allow for the reconstruction, in a particularly effective way, of the approximate rules relating to a set of "explanatory" data concerning the considered problem (the Input), with a set of data (the Output) for which it is requested to make a correct forecast or reproduction in conditions of incomplete information.
– Evolutionary Systems (ES): the generation of adaptive systems changing their architecture and their functions over time in order to adapt to the environment into which they are integrated, or comply with the links and rules that define their environment and, therefore, the problem to be simulated. Basically, these are systems that are developed to find data and/or optimum rules within the statically and dynamically determined links and/or rules. The development of a genotype from a time ti to a time t (i+n) is a good example of the development over time of the architecture and functions of an adaptive system.

2. A brief introduction to Artificial Neural Networks

2.1 *Architecture*

ANNs are a family of methods inspired to the human brain learning capability.

ANNs are scientifically used in three different epistemological directions:

1) To understand the working of human brain, by simulation;
2) To optimize parallel computation research (human brain emulation);
3) To understand the transition from individual to collective behavior (Data Analysis, Data Mining and the research on Complex Systems are part of this point).

Currently ANNs comprise a range of very different models, but they all share the following characteristics:

1) The fundamental elements of each ANN are the Nodes, also known as Processing Elements (PE), and their Connections.
2) Each node in an ANN has its own Input, through which it receives communications from the other nodes or from the environment; and its own Output, through which it communicates with other nodes or with the en-

vironment. Finally it has a function, f(•), by which it transforms its global Input into Output.
3) Each Connection is characterized by the force with which the pair of nodes excites or inhibits each other: positive values indicate excitatory connections and negative ones indicate inhibitory connections.
4) Connections between nodes may change over time. This dynamic triggers a learning process throughout the entire ANN. The way (the law by which) the connections change in time is called the "Learning Equation".
5) The overall dynamic of an ANN is linked to time: in order for the connections of the ANN to properly change, the environment must act on the ANN several times.
6) When ANNs are used to process data, these latter are their environment. Thus, in order to process data, these latter data must be subjected to the ANN several times.
7) The overall dynamic of an ANN depends exclusively on the local interaction of its nodes. The final state of the ANN must, therefore, evolve spontaneously from the interaction of all of its components (nodes).
8) Communications between nodes in every ANN tend to occur in parallel. This parallelism may be synchronous or asynchronous and each ANN may emphasize it in a different way. However, an ANN must present some form of parallelism in the activity of its nodes.

From a theoretical viewpoint this parallelism does not depend on the hardware on which the ANNs are implemented. Every ANN must present the following architectural components:

1) Type and number of nodes and their corresponding properties;
2) Type and number of connections and their corresponding location;
3) Type of Signal Flow Strategy;
4) Type of Learning Strategy.

2.2 *The nodes*

There can be three types of ANN nodes, depending on the position they occupy within the ANN:

1) Input nodes: the nodes that (also) receive signals from the environment outside the ANN.
2) Output nodes: the nodes whose signal (also) acts on the environment outside the ANN.
3) Hidden nodes: the nodes that receive signals only from other nodes in the ANN and send their signal only to other nodes in the ANN.

The number of Input nodes depends on the way the ANN is intended to read the environment. The Input nodes are the ANN's sensors. When the

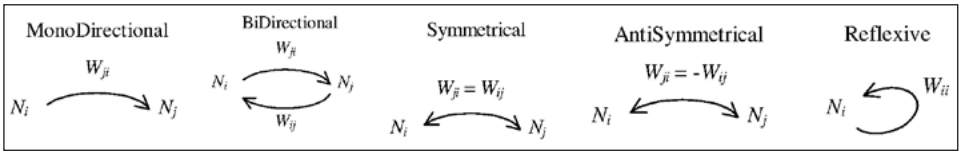

Fig. 5 – Types of possible connections.

ANN's environment consists of data the ANN should process, the Input node corresponds to a sort of data variable. The number of Output nodes depends on the way one wants the ANN to act on the environment. The Output nodes are the effectors of the ANN. When the ANN's environment consists of data to process, the Output nodes represent the variables sought or the results of the processing that occurs within the ANN. The number of hidden nodes depends on the complexity of the function one intends to map between the Input nodes and the Output nodes. The nodes of each ANN may be grouped into classes of nodes sharing the same properties. Normally these classes are called layers. Various types can be distinguished:

1) Monolayer ANNs: all nodes of the ANN have the same properties.
2) Multilayer ANNs: the ANN nodes are grouped in functional classes; for example, nodes that (a) share the same signal transfer functions or (b) receive the signal only from nodes of other layers and send them only to new layers.
3) Nodes Sensitive ANNs: each node is specific to the position it occupies within the ANN; e.g. the nodes closest together communicate more intensely than they do with those further away.

2.3 *The connections*

There may be various types of connections: Mono-Directional, Bi-directional, Symmetrical, Anti-Symmetrical and Reflexive (Fig. 5). The number of connections is proportional to the memory capabilities of the ANN. Positioning the connections may be useful as methodological preprocessing for the problem to be solved, but it is not necessary. An ANN in which the connections between nodes or between layers are not all enabled is called an ANN with Dedicated Connections; otherwise it is known as a maximum gradient ANN. In each ANN the connections may be:

1) Adaptive: they change depending on the learning equation.
2) Fixed: they remain at fixed values throughout the learning process.
3) Variable: they change deterministically as other connections change.

2.4 *The signal flow*

In every ANN the signal may proceed in a direct fashion (from Input to Output) or in a complex fashion. Thus we have two types of Flow Strategy:

1) Feed forward ANN: the signal proceeds from the Input to the Output of the ANN passing all nodes only once.
2) ANN with Feedback: the signal proceeds with specific feedbacks, determined beforehand, or depending on the presence of particular conditions.

The ANNs with Feedback are also known as Recurrent ANNs, and are the most plausible from a biological point of view. They are often used to process timing signals and they are the most complex to deal with mathematically. In an industrial context, therefore, they are often used with feedback conditions determined *a priori* (in order to ensure stability).

3. Learning in the Artificial Neural Network

Every ANN can learn, over some period of time, the properties of the environment in which it is immersed or the characteristics of the data which it presents. This is accomplished in basically one of two ways (or mixture of both):

1) By reconstructing approximately the probability density function of the data received from the environment, compared with preset constraints.
2) By reconstructing approximately the parameters which solve the equation relating the Input data to the Output data, compared with preset constraints.

The first method is known in the context of ANNs as Vector Quantization; the second method is Gradient Descent. The Vector Quantization method articulates the Input and Output variables in hyperspheres of a defined range. The Gradient Descent method articulates the Input and Output variables in hyperplanes. The difference between these two methods becomes evident in the case of a feed forward ANN with at least one hidden unit layer. With Vector Quantization the hidden units encode locally the more relevant traits of the Input vector. At the end of the learning process, each hidden unit will be a Prototype representing one or more relevant traits of the Input vector in definitive and exclusive form. With Gradient Descent, the hidden units encode in a distributed manner the most relevant characteristics of the Input vector. At the end of the learning process, each hidden unit will tend to represent the relevant traits of the Input in a fuzzy and non-exclusive fashion. Summing up, the Vector Quantization develops a local learning, and the Gradient Descent develops a distributed or vectorial learning. Considerable differences exist between the two approaches:

1) Distributed learning is computationally more efficient than local learning. It may also be more biologically plausible (not always nor in every case).
2) When the function that connects Input to Output is nonlinear, distributed learning may "jam" on local minimums due to the use of the Gradient Descent technique.
3) Local learning is often quicker than distributed learning.
4) The regionalization of Input on Output is more sharply defined when using Vector Quantization than when using Gradient Descent.
5) When interrogating an ANN trained with Vector Quantization, the ANN responses cannot be different from those given during learning; in the case of an ANN trained with Gradient Descent the responses may be different from those obtained during the learning phase.
6) This feature is so important that families of ANNs treating the signal in 2 steps have been designed: first with the Quantization method and then with the Gradient method.
7) Local learning helps the researcher to understand how the ANN has interpreted and solved the problem; distributed learning makes this task more complicated (though not impossible).
8) Local learning is a competitive type; distributed learning presents aspects of both competitive and cooperative behavior between the nodes.

4. Artificial Neural Network typology

Traditionally ANNs are divided into two families: Supervised ANNs and Unsupervised ANNs. But from a theoretical point of view this distinction could be superficial. An interesting viewpoint on this theoretical debate can be gained by noting that, from the point of view of the energy function that is being calculated by an unsupervised vs. a supervised ANN, it is easy to subsume both approaches into a common framework. The energy function for a supervised ANN can be written as its Mean Square Error:

$$MSE = \frac{1}{2}\sum_{p}^{K}\sum_{i}^{N}\left(t_{p,i} - u_{p,i}\right)^2 \tag{1}$$

Whereas, traditionally, the energy minimization function in an unsupervised auto-associative neural network is represented by the following equation:

$$En = \frac{1}{2}\sum_{p}^{K}\sum_{i}^{N}\sum_{j}^{n}\left(u_{p,i} \cdot u_{p,j} \cdot w_{i,j}\right) \tag{2}$$

where $w_{i,j}$ = trained weights from Input j to Input i.

But if we assume that equation (1) represents the mean error of a linear perceptron, then we can develop equation (1) as follows:

$$MSE = \frac{1}{2}\sum_{p}^{K}\sum_{i}^{N}\left(t_{p,i} - u_{p,i}\right)^2 = \frac{1}{2}\cdot\sum_{p}^{K}\sum_{i}^{N}\left(t_{p,i} - \sum_{j}^{N} u_{p,i}\cdot w_{i,j}\right)^2 = \frac{1}{2}\sum_{p}^{K}\sum_{i}^{N}\left(t_{p,i}^2 - 2\cdot t_{p,i}\cdot\sum_{j}^{N} u_{p,j}\cdot w_{i,j} + \left(\sum_{j}^{N} u_{p,i}\cdot w_{i,j}\right)^2\right) \tag{3}$$

Setting all targets to 0, as in the case of unsupervised neural networks, we have:

$$MSE = \frac{1}{2}\sum_{p}^{K}\sum_{i}^{N}\left(\sum_{j}^{N} u_{p,j}\cdot w_{i,j}\right)^2 = \frac{1}{2}\sum_{p}^{K}\sum_{i}^{N}\left(\sum_{j}^{N} u_{p,j}\cdot w_{i,j}\right)\left(\sum_{j}^{N} u_{p,j}\cdot w_{i,j}\right) = \frac{1}{2}\sum_{p}^{K}\sum_{i}^{N} u_{p,i}\left(\sum_{j}^{N} u_{p,j}\cdot w_{i,j}\right) \tag{4}$$

At this point it is easy to derive:

$$MSE = \frac{1}{2}\sum_{p}^{K}\sum_{i}^{N}\sum_{j}^{N} u_{p,i}\cdot u_{p,j}\cdot w_{i,j} \tag{5}$$

and equation (5) is the energy function for an unsupervised ANN (see equation (2).

Therefore:

$$En = MSE\ when\ (target = 0) \tag{6}$$

We can thus in principle regard unsupervised ANN learning as a conceptually more economical approach than supervised learning in that it entails doing away with some free parameters, namely, targets. Or, on the other hand, we can make a case for supervised learning, i.e. for the inclusion of the extra free parameters, as a way to focus the learning model upon a more clear-cut task. Adopting this point of view ANNs can be classified into three sub families:

1) Supervised ANNs;
2) Unsupervised Auto Associative Memories;
3) Unsupervised Autopoietic ANNs.

Furthermore, a unique pseudo code can be used as general framework to build any kind of ANN (Supervised and Unsupervised):

1. Design of the Architecture of the Network
2. Initialization of Weights
3. Do Epochs
{
do Cycles
{
a. Presentation of one Pattern as Input Vector
b. Signal Transfer up to the Output layer
c. Error Computation for each Node and Weight
d. Weights and/or Nodes updating
e. Possible Recurrence
} Until (all Patterns are presented)
} Until (Cost Function is Optimized)

At this point we can describe from a didactic standpoint three families of different problem and consequently the three more famous families of ANNs.

4.1 *Supervised ANNs*

The first type of problem with which an ANN can deal is expressed as follows: given N variables, about which it is easy to gather data, and M variables, which differ from the first and about which it is difficult and costly to gather data, assess whether it is possible to predict the values of the M variables on the basis of the N variables. This family of ANNs is named Supervised ANNs (SV) and their prototypical equation is:

$$y = f(x, w^*) \tag{7}$$

where y is the vector of the M variables to predict and/or to recognize (target), x is the vector of N variables working as networks Inputs, w is the set of parameters to approximate and f ( ) is a non-linear and composed function to model. When the M variables occur in time subsequent to the N variables, the problem is described as a prediction problem; when the M variables depend on some sort of typology, the problem is described as one of recognition and/or classification (this is also sometimes referred to as the proscription problem).

Conceptually it is the same kind of problem: using values for some known variables to predict the values of other unknown variables. In order to correctly apply an ANN to this type of problem we need to run a validation

protocol. We must start with a good sample of cases, in each of which the N variables (known) and the M variables (to be discovered) are both known and reliable. The sample of complete data is needed in order to:

1) train the ANN, and
2) assess its predictive performance.

The validation protocol uses part of the sample to train the ANN (Training Set), whilst the remaining cases are used to assess the predictive capability of the ANN (Testing Set or Validation Set). In this way we are able to test the reliability of the ANN in tackling the problem before putting it into operation. Now we provide some example of Supervised ANNs.

4.2 *The Multi-Layer Perceptron*

Back Propagation (BP for short) refers to a broad family of Multi-Layer Perceptron, whose architecture consists of different interconnected layers [1-4]. The BP ANNs represents a kind of supervised ANN, whose learning's algorithm is based on the Deepest-Descent technique. If provided with an appropriate number of Hidden units, they will also be able to minimize the error of nonlinear functions of high complexity (Fig. 6).

Theoretically, a BP provided with a simple layer of Hidden units is sufficient to map any function $y = f(x)$. Practically, it is often necessary to provide these ANNs with at least 2 layers of Hidden units, when the function to compute is particularly complex, or when the chosen data, in order to train the BP, are not particularly reliable, and a level filter is necessary on the features of Input. The BP are networks, whose learning function tends to "distribute itself" on the connections, just for the specific correction algorithm of the weights that is utilized. This means that, in the case of BP, provided with at least one layer of Hidden units, these units tend to distribute among themselves the codification of each feature of the Input vector.

This makes the learning more compact and efficient, but it is more complex to know the "reasoning" which brings a BP, in the testing process, to answer in a certain way. In brief, it is difficult to explain the implicit knowledge acquired by these ANNs in the training process.

A second theoretical and operative difficulty raised by BP concerns the minimum number of Hidden units that are necessary for these ANNs in order to compute a function. In fact, it is known that if the function is not linear, at least one layer of Hidden units will be necessary. But, at the moment, the exact minimum number of Hidden units needed to compute a non-linear function is unknown. In these cases, we base our work on experience and on some heuristics.

Experience advises us to use a minimum number of Hidden units in a first time training of an ANN. If the training succeeds, an analysis of the

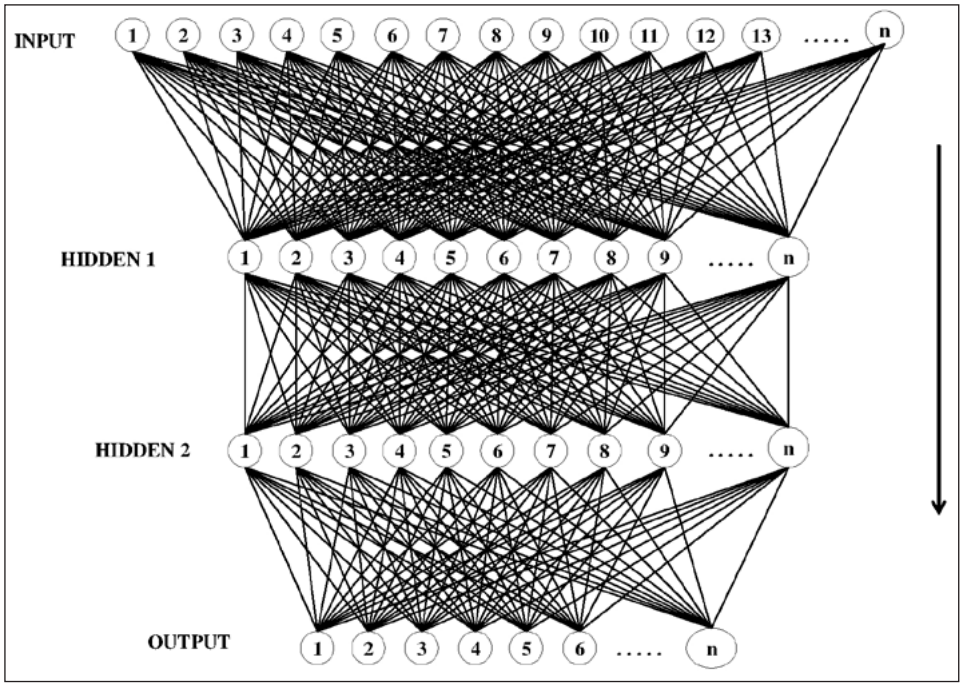

Fig. 6 – Example of Multi-Layer Perceptron.

sensitivity will normally allow us to understand the singularity number that each Input node determines on the Output, and, consequently, it will be able to deduce the degree of freedom needed by the ANN to resolve the equation, and then to express these latter under the form of Hidden units. This procedure is not guaranteed; during the training process the BP can become trapped in local's minima.

This is because of the relation between the morphology complexity of the hyper surface that characterizes the function and the weights' values, randomly set and placed before the training.

The dilemma of BP is that for a prior, unknown minimum number of Hidden units useful to compute a function, if too many are created, the BP can create during some forms of training a condition of over fitting, which causes a worsening of its generalization capacities in the testing process. If not too many are created, the BP can have difficulty learning either because the function is too complex, or because the BP randomly falls into a local minimum. The BP's family includes both Feed Forward ANN and Feedback ANN, also known as Recurrent Networks (Chauvin, Rumelhart 1995).

4.3 *The Conic Net*

The Conic Net is a supervised ANN, designed by P.M. Buscema in 2013 and never published before. The architecture is similar to that of a Multilayer Perceptron, but its hidden layer is completely different. Each traditional node of the classic Hidden layer in the case of the Conic Net (CN) is decomposed in 3 sub nodes connected by 6 weights, according to Fig. 7a. This topology aims to transform each complex hidden node into a quadratics equation, whose parameters have to be learned during the training phase (Fig. 7b).

The singularity of Conic Net in relation to the other and more classic MLP is its complex hidden layer structure: two sub nodes receiving their weights vectors independently from the same Input vector and one sub node working as Output node, receiving the 6 parameters from the quadratics equation, including the two previous sub nodes as X and Y of the conic function. Further, it is interesting to note that the two X and Y sub nodes modify their incoming weights each one according to two different and independent learning laws: the gradient descent (the X sub node) and the quantization algorithm (the Y sub node).

The following equations show how the signals flow from Input layer to Output layer in the Conic Net:

$$Net_i = \sum_j^N u_j \cdot w(x)_{i,j} + \theta_i \tag{8}$$

$$Qx_i = \frac{1}{1 + e^{-Net_i}} \tag{9}$$

$$D_i = \sum_j^N (N \cdot (2 \cdot u_i - 1) - w(y)_{i,j})^2 \tag{10}$$

$$Qy_i = \left(1 - \frac{\sqrt{D_i}}{N}\right) \cdot Exp\left(-\frac{D_i}{N}\right) \tag{11}$$

$$C_i = aQ \cdot x^2\ 2bQxQx + cQ \cdot y^2 + 2dQx + 2eQy + f \tag{12}$$

$$Qz_i = \frac{1}{1 + e^{-C_i}} \tag{13}$$

where:

u = Input Vector
N = Number of Inputs
Qx = Output of the X sub node of the CN hidden layer
Qy = Output of the Y sub node of the CN hidden layer

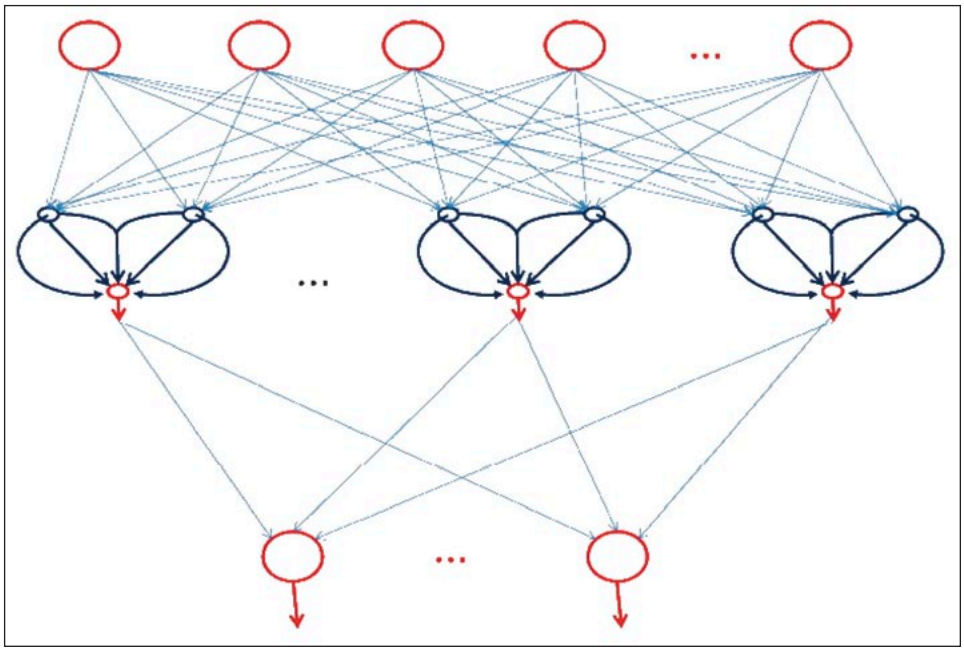

Fig. 7a – Topology of Conic Net (in blue the Hidden layer).

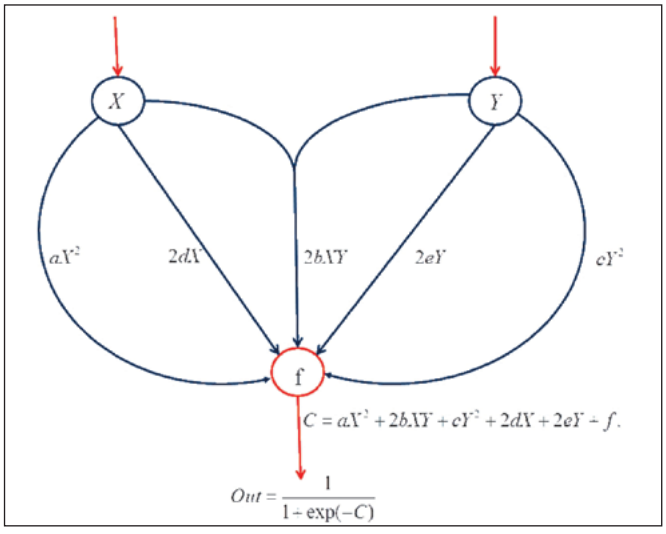

Fig. 7b – Close up of one macro hidden node of a Conic Net.

C = Output of the third sub node of the CN hidden layer

The following equations show how we calculate the local error in CN:

$$\delta_i^{[n]} = \sum_{j=1}^{N} \delta_j^{[n+1]} \cdot w_{ji}^{[n+1]} \tag{14}$$

$$\Delta w(x)_{i,j}^{[n]} = \delta_i^{[n]} \cdot f'(Qx_i) \cdot u_j^{[n-1]} \cdot \varepsilon \tag{15}$$

$$\Delta w(y)_{i,j}^{[n]} = \delta_i^{[n]} \cdot (Qy_i) \cdot \left(N \cdot (2 \cdot u_j^{[n-1]} - 1 - w(Qy)_{i,j}\right) \cdot \varepsilon \tag{16}$$

$$\Delta w_a = \delta_i^{[n]} \cdot Qx_i^2 \cdot \varepsilon \tag{17}$$

$$\Delta w_b = \delta_i^{[n]} \cdot 2 \cdot Qx_i \cdot Qx_y \cdot \varepsilon \tag{18}$$

$$\Delta w_c = \delta_i^{[n]} \cdot 2 \cdot Qy^2 \cdot \varepsilon \tag{19}$$

$$\Delta w_d = \delta_i^{[n]} \cdot 2 \cdot Qx_i \cdot \varepsilon \tag{20}$$

$$\Delta w_e = \delta_i^{[n]} \cdot 2 \cdot Qx_y \cdot \varepsilon \tag{21}$$

$$\Delta w_f = \delta_i^{[n]} \cdot \varepsilon \tag{22}$$

where**:**

$\delta_j^{[n+1]}$ = error of the next layer calculated BP delta rule

$w_{ji}^{[n+1]}$ = the weights matrix of the next layer

And, finally, the equations by means we correct the Input weights and the quadratics weights (parameters) of the CN:

$$w(x)_{i,j}(t+1) = w(x)_{i,j}(t) \cdot \Delta w(x)_{i,j}^{[n]} \tag{23}$$

$$w(y)_{i,j}(t+1) = w(y)_{i,j}(t) \cdot \Delta w(y)_{i,j}^{[n]} \tag{24}$$

$$w_a(t+1) = w_a(t) + w\Delta_{\square} \tag{25}$$

$$w_b(t+1) = w_b(t) + w\Delta_b \tag{26}$$

$$w_c(t+1) = w_c(t) + w\Delta_c \tag{27}$$

$$w_d(t+1) = w_d(t) + w\Delta_d \tag{28}$$

$$w_e(t+1) = w_e(t) + w\Delta_e \tag{29}$$

$$w_f(t+1) = w_f(t) + w\Delta_f \tag{30}$$

The CN presents also many suitable features that in this paper are not pertinent to describe into details.

4.4 *The Supervised Contractive Map*

The Supervised Contractive Map (SVCM for short) was designed by M. Buscema in 1999 (Buscema, Benzi 2011). This ANN calculates two net Inputs for each node: a classic weighted Input (see Equation 31) and a contractive Input (see Equation 32). This second net Input tends to decay or to increase when the positive or negative value of the weight (w) becomes close to a specific constant (C). Equation 33 activates each node according to a sine function of the two net Inputs (the contractive Input works as a harmonic modulation of the weighted Input). The vantages and the disadvantages of sine transfer function to work properly into the topology of Multilayer Perceptron were already analyzed in the scientific literature (Le Cun *et al.* 1991).

$$CNet_i^{[l]} = \sum_j^{C^{[l-1]}} u_j^{[l-1]} \cdot \left(1 - \frac{w_{i,j}^{[l]}}{C^{[l-1]}}\right) \tag{31}$$

$$INet_i^{[l]} = \sum_j^{C^{[l-1]}} u_j^{[l-1]} \cdot w_{i,j}^{[l]} \tag{32}$$

$$u_i^{[l]} = sin\left(INet_i^{[l]} \cdot \left(1 - \frac{\sin(CNet_i^{[l]})}{C^{[l-1]}}\right)\right) \tag{33}$$

Equation 34 shows a typical error calculation using the distance between the desiderate Output and the estimated Output, times the first derivative of sine transfer function. Equation 35 works in the same way of Equation 34, but using the chain rule to calculate the local error of each hidden unit. Equation 36 updates the weight matrices, using typical back error propagation, with a contractive factor useful to limit an extreme growing of each weight value.

$$\delta_i^{[out]} = (t_i - u_i^{out}) \cdot \cos\left(INet_i^{[out]} \cdot \left(1 - \frac{\sin\left(CNet_i^{[out]}\right)}{C^{[out-1]}}\right)\right) \tag{34}$$

$$\delta_i^{[hid]} = \sum_k^{Num[hid+1]} (\delta_k^{[k+1]} \cdot w_{k,i}^{[hid+1]}) \cdot \cos\left(INet_i^{[hid]} \cdot \left(1 - \frac{\sin\left(CNet_i^{[hid]} \right)}{C^{[hid-1]}} \right) \right) \tag{35}$$

$$\Delta w_{i,j}^{[l]} = LCoef \cdot \delta_i^{[l]} \cdot u_j^{[l-1]} \cdot \left(1 - \frac{w_{i,j}^{[l]}}{C^{[l-1]}} \right) \tag{36}$$

where:

[l] = number or name of the ANN layer;
$u_j^{[l]}$ = values of the all i-th nodes of the l-th layer;
$w_{i,j}^{[l]:}$ = weight matrix connecting the layer [l-1] to the layer [l];
$C^{[l]}$: number of nodes of the l-th layer;
t_i: = value of the i-th of the dependent variable;
LCoef = ANN learning rate.

The SVCM is been already tested for the approximation of highly non-linear and complex interpolation with excellent results (Buscema, Benzi 2011).

4.5 *Dynamic Associative Memories*

The second type of problem that an ANN raises can be expressed as follows: given N variables defining a dataset, find out its optimal connections matrix able to define each variable in terms of the others and consequently to approximate the hyper-surface on which each data-point is located.

This second sub-family of ANNs is named Dynamic Associative Memories (DAM). The specificity of these ANNs is incomplete pattern reconstruction, dynamic scenario simulation and possible situations prototyping. Their representative equation is:

$$x^{[n+1]} = f(x^n, w^*) \tag{37}$$

where x[n] is the N variables evolving in the ANNs internal time, w* is the connection matrix approximating the parameters of the hyper-surface representing the dataset, and f() is some suitable non-linear and eventually composed function governing the process. DAM ANNs after the training phase need to be submitted to a validation protocol named "Data Reconstruction Blind Test". In this test the capability of a DAM ANN to rebuild complete data from uncompleted ones is evaluated from a quantitative point of view. Now we describe briefly some type of Auto-Associative Memory ANN.

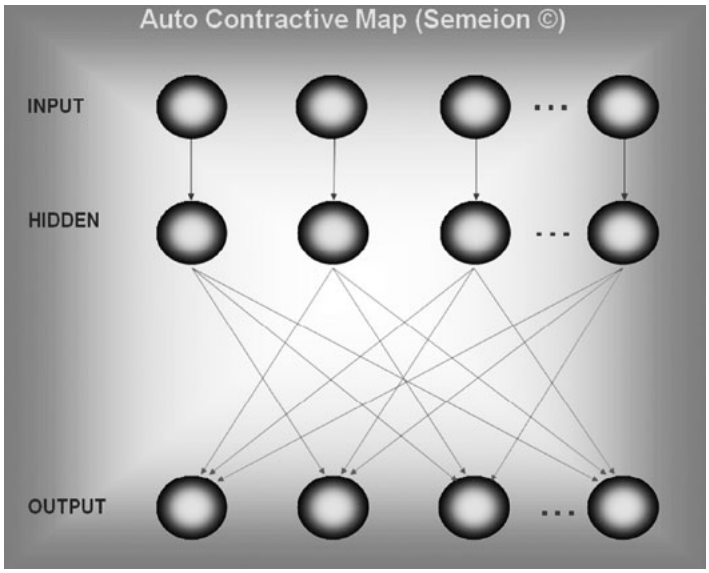

Fig. 8 – AutoContractive Map.

4.6 *AutoContractive Map*

The AutoContractive Map neural network (Auto-CM for short) was designed by P. M. Buscema in 1999 and its learning law was improved up to 2013. Auto-CM was explained in different papers and it was applied in many fields with promising results (Buscema 2007a, 2007b; Buscema *et al.* 2008a, 2008b, 2010; Buscema, Sacco 2010; Licastro *et al.* 2010; Grossi *et al.* 2011). The software implementing Auto-CM is developed by Semeion Research Center in Rome and it is available free for academic applications. Auto-CM has an architecture based on three layers of nodes: an Input layer that captures the signal from the environment, a hidden layer which modulates the signal within the network, and an Output layer which returns a response to the environment on the basis of the processing that occurred. The three layers have the same number N of nodes.

The connections between the Input layer and the hidden one are mono-dedicated, whereas those between this hidden layer and the Output layer are completely connected. Each connection is assigned a weight: v_i for connections between the i^{th} Input node and the corresponding hidden node, $w_{i,j}$ for those between the generic j^{th} node of the hidden layer and the i^{th} node of the Output layer (Fig. 8). For the training, datasets are scaled between zero and one and all weights are initialized beforehand to the same positive value close to zero.

Symbol	***Meaning***
x_i^p	*i^{th} input node of the p^{th} pattern*
$h_i^p(n)$	*i^{th} hidden node of the p^{th} pattern during the n^{th} time*
$y_i^p(n)$	*i^{th} node in the output of the p^{th} pattern during the n^{th} epoch*
$v_i(n)$	*weight of the connection between the i^{th} input node and in the i^{th} hidden node during the n^{th} epoch*
$w_{i,j}(n)$	*weight of the connection between the j^{th} hidden node and the i^{th} output node during the n^{th} epoch*
N	*the number of nodes per layer*
M	*the number of patterns*
α	*constant learning rate*
C	*constant greater than one, typically* $C = \sqrt{N}$

Tab. 1 – Notation for Auto-CM Neural Network.

Then the network must undergo a series of epochs. In each of them, all the Input patterns must be presented one after another to the network, and a calculation made for the appropriate equations with the corresponding Output value and a measure of error with respect to the desired value. In accordance with the principle of batch update, the corrections accumulated for an epoch must be applied at the end. The equations of training of the network make reference to the quantities shown below (Tab. 1).

At the n^{th} epoch of training, out of each Input pattern a value is calculated for the hidden layer, through a contraction, that reduces the Input value in proportion to the mono-dedicated weight.

$$h_i^{[p]}(n) = x_i^{[p]} \cdot (1 - \frac{v_i(n)}{C}) \tag{38}$$

The algorithm then calculates the value on the Output layer through a "double conceptual passage". For each Output node, an initial operation saves the net Input calculation, that is to say, the reduction (contraction) of all the hidden nodes through the weights between the hidden layer and Output layer (Equation 39).

$$Net_i^{[p]}(n) = \sum_{j=1}^{N} h_i^{[p]}(n) \cdot (1 - \frac{v_i(n)}{C}) \tag{39}$$

A second operation calculates the Output value by further contracting the corresponding value of the hidden node thorough the previously calculated net Input for the Output node:

$$y_i^{[p]}(n) = h_i^{[p]}(n) \cdot (1 - \frac{Net_i^{[p]}(n)}{C} \tag{40}$$

During the training that occurs in every epoch, in addition to the calculation of the Output values (40), for each pattern presented in Input the algorithm calculates the correction quantity of the weights, summed and applied at the end of the epoch. For the N-mono dedicated layers between the Input and hidden layers, the algorithm considers the contraction, based on the weight being examined, of the difference between the values of the corresponding Input and hidden nodes, further modulated for the Input node itself.

$$\Delta v_i(n) = \sum_{p=1}^{M} (x_i^{[p]} - h_i^{[p]}(n)) \cdot (1 - \frac{v_i(n)}{C}) \tag{41}$$

$$v_i(n+1) = v_i(n) + \alpha \cdot \Delta v_i(n) \tag{42}$$

Similarly, for N2 weights between the hidden and Output layers the algorithm calculates the contraction, based on the weight being considered, between the corresponding hidden and Output nodes.

$$\Delta w_i(n) = \sum_{p=1}^{M} (h_i^{[p]} - y_i^{[p]}(n)) \cdot (1 - \frac{w_{i,j}(n)}{C}) \cdot h_i^{[p]}(n) \tag{43}$$

$$w_{i,j}(n+1) = w_{i,j}(n) + \alpha \cdot \Delta w_{i,j}(n) \tag{44}$$

The equations immediately illustrate how the contractions establish a relationship of order between the layers:

$$x_i^{[p]} \geq h_i^{[p]}(n) \geq y_i^{[p]}(n) \tag{45}$$

As we can easily observe during the training, the mono-dedicated weights vi grow monotonically, and with different speeds asymptotically towards the constant C:

$$\lim_{x \to \infty} \Delta w_i(n) = 0 \tag{46}$$

$$\lim_{x \to \infty} v_i(n) = C \tag{47}$$

just like the values of hidden nodes tend to cancel themselves out:

$$\lim_{x\to\infty} h_i^{[p]}(n) = 0 \tag{48}$$

along with those of the Output units:

$$\lim_{x\to\infty} y_i^{[p]}(n) = 0 \tag{49}$$

while the corrections of the full set of weights diminish:

$$\lim_{x\to\infty} \Delta w_{i,j}(n) = 0 \tag{50}$$

The process of cancelling the above quantity occurs with speed modulated by the Input patterns and leaves its specific sign in the matrix between the hidden and the Output layer.

4.7 *Autopoietic ANNs*

The third type of ANNs can be described as follows: given N variables defining M records in a dataset, evaluate how these variables are distributed and how these records are naturally clustered in a small projection space K (K<<N) according to their most important relationships. These ANNs are named Autopoietic ANNs. Their specificity is the nonlinear extraction of the similarities among records in a database, using all the variables at the same time.

One important feature of these ANNs is also the possibility that some of them have to visualize in a 2 or 3 dimensional map the geographical similarities among records and among variables. The prototypical equation of the Autopoietic ANNs is:

$$y^{[n+1]} = f(y^{[n]}, x, w^*) \tag{51}$$

where y is the projection result along the time, x is the Input vector (independent variables) and w is the set of parameters (codebooks) to be approximated. In Autopoietic ANNs, the codebooks (w) after the training phase represent an interesting case of cognitive abstraction: in each codebook the ANN tends to develop its abstract cognitive representation of some of the data which it learnt. Self-Organizing Map (SOM) is a known example of Autopoietic ANN.

4.8 *Self-Organizing Map (SOM)*

The Self-Organizing Map (SOM) is a neural network attributed to Teuvo Kohonen (1982, 1984, 1990, 1995), who developed it between 1979 and 1982. It is an unsupervised type of network which offers a classification of the Input vectors creating a prototype of the classes and a projection of the proto-

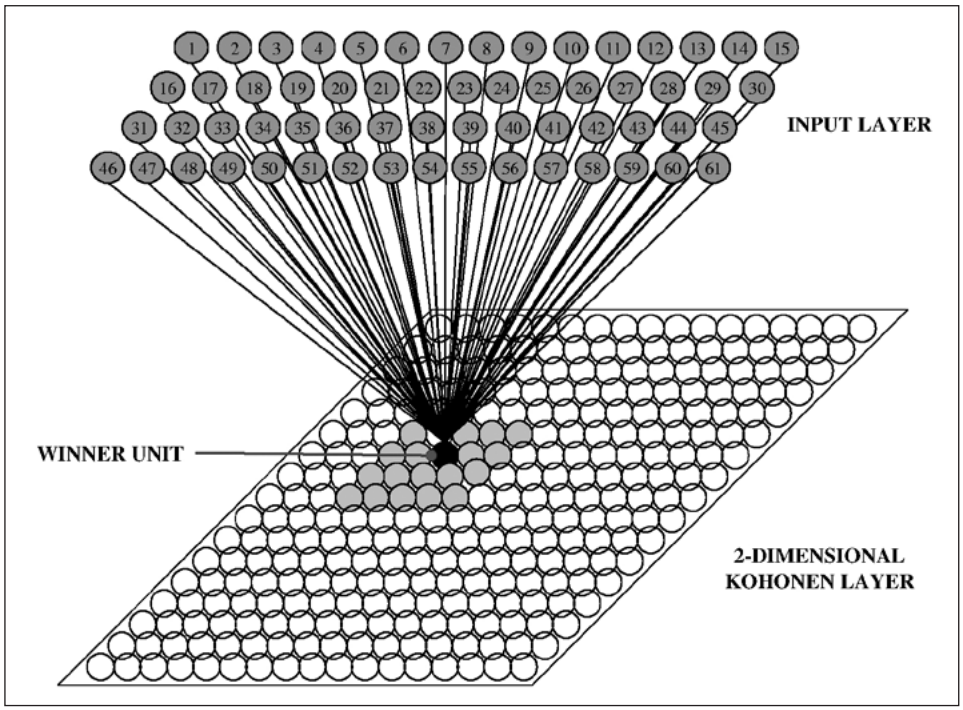

Fig. 9 – Example of Unsupervised ANN for natural clustering – Self-Organizing Map.

types on a two-dimensional map (but n-dimensional maps are also possible) able to record the relative proximity (or neighborhood) between the classes. Therefore, the network offers important synthetic information on the Input:

1) It operates a classification of the Input vectors on the basis of their vector similarity and assigns them to a class;
2) It creates a prototypical model of the classes with the same cardinality (number of variables) as the Input vector;
3) It provides a measurement, expressed as a numerical value, of the distance/proximity of the various classes;
d. It creates a relational map of the various classes, placing each class on the map itself;
4) It provides a measurement of the distance/proximity existing between the Input vectors from the class they belong to and between the Input vectors and other classes.

The relative simplicity of the network architecture allowed its dissemination in terms of how successfully its implementation could be replicated (Fig.

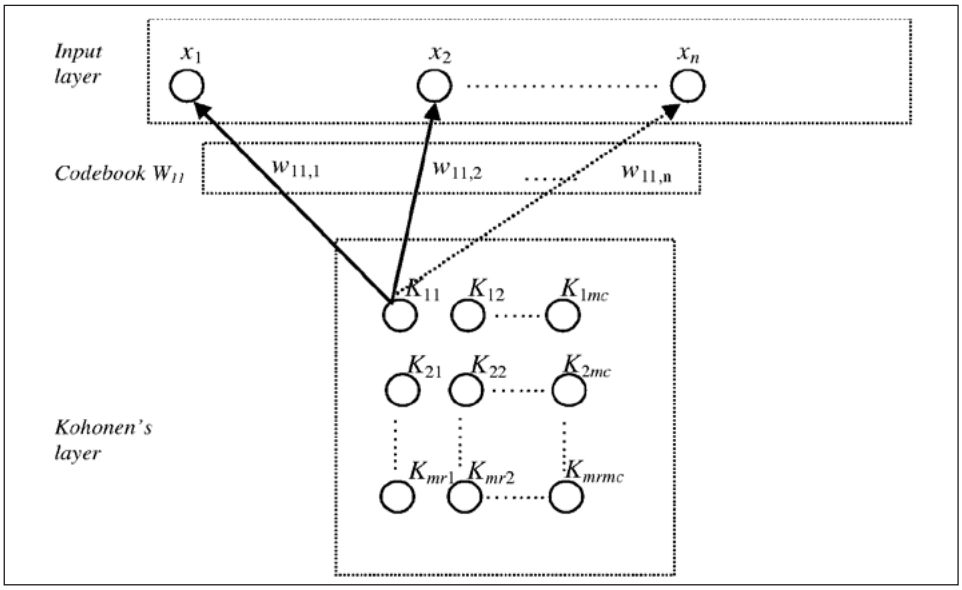

Fig. 10 – SOM with n-nodes of Input, with (mrmc) units of Kohonen's layer. This architecture allows the Inputs to be classified into m2 classes, each being a subclass represented by a codebook.

9). A typical SOM network is made up of 2 layers of units: a one-dimensional Input (n-cardinality vector) and a two-dimensional Output layer (lines (r) × columns (c)), also known as Kohonen's map (M matrix of mr x mc dimensions). A matrix of the weights records the relation between each unit of the Output layer and each unit of the Input layer (W matrix of (mr × mc × n) dimensions). The weight vector connecting each Output unit to an Input unit is called a "code-book" (vector wrc of n-cardinality) (Fig. 10).

Within the SOM network each Output unit can be interpreted as a class whose codebook represents the prototype. The SOM algorithm is based on a competitive algorithm founded on the vector quantification principle: at each cycle of life in the network, the unit from Kohonen's layer whose codebook is most similar to the Input wins. This unit is given the name of Winner Unit (WU). Consequently, the WU codebook is modified to get it even closer to the Input. The codebooks belonging to the units that are physically near the WU (which are part of the neighborhood) are also put closer to the Input of a given delta.

The algorithm calculates a first stage during which the parameters of neighborhood and corrections of weights are set and the codebook initialization is carried out; this stage is followed by the cyclic stage of codebook adjustment. In this stage the codebooks are modified for the network to classify the Input records. In short, the SOM algorithm is organized as follows:

Initialization stage:

1) Layering of the Input vectors;
2) Definition of the dimensions (rows x columns) of the matrix which, in its turn, determines the number of classes and therefore of prototypes (codebook);
3)Initialization of the codebooks: the values of the vectors of each codebook are random;
4) Definition of the function (Gaussian, Mexican hat, etc.) and of the parameters regulating the neighborhood of the Winner Unit and of the weight correction delta.

Cyclic calibration stage:

1) Presentation of the Input vectors (pattern) in a random and cyclic way.
2) Calculation of the d-activation of the *K* units of Kohonen's layer: the activation is calculated as vector distance between the Input vector X and the weight vector *Wj* (mj codebook) which links the *K* unit to the Input nodes.

The classic way to calculate the Euclidean distance between the vectors is:

$$d_j = \|X - W_j\| = \sqrt{\sum_{i=1}^{N} (x_i - w_{i,j})^2} \tag{52}$$

3) Determination of the winning unit WU: the node of the K layer whose activation is less:

$$WU = d_w = \min_{j \in [1,M]} \left\{ d_j \|X - W_j\| = \sqrt{\sum_{i=1}^{N} (x_i - w_{i,j})^2} \right\} \tag{53}$$

4) Correction of the codebook (matrix of the W_{ij} weights) of the winning unit and the units adjacent to the winning unit in relation to the function set to determine the level of weight correction according to the Input and the proximity to the *WU*.
5) Updating of the factors determining the proximity and layering of the delta correction of the codebooks.

The distinctive characteristic of the SOM is mainly related to the updating of the weights, carried out not only on those related to the WU but also, according to the chosen function, on the weights be-longing to the units which are physically close to it.

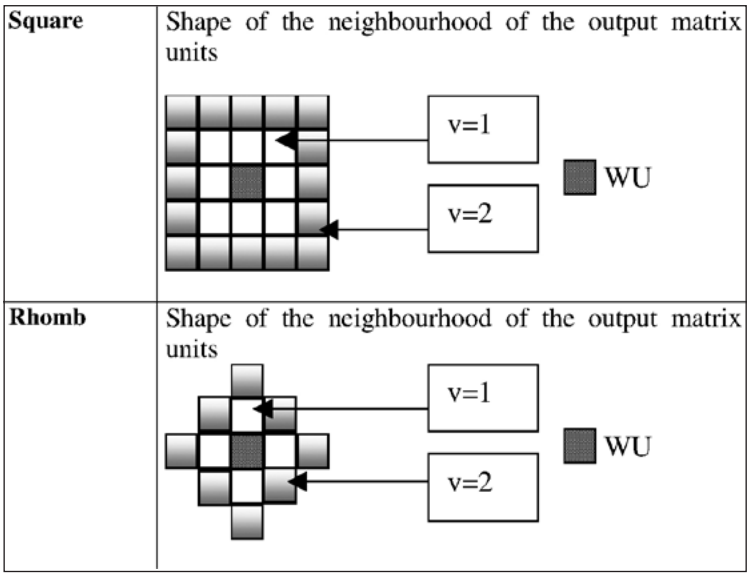

Fig. 11 – Topology of the neighborhood Space of a Winner Unit in a square and in a rhomb; in the illustration v is the degree of proximity of the K units to the WU.

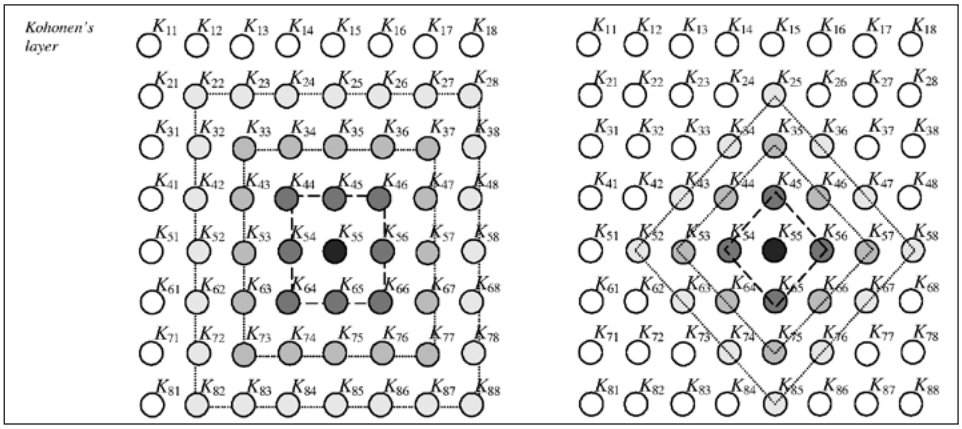

Fig. 12 – Example of the topology of the neighborhood space with matrix K (8r × 8c), where the WU is the K55 unit. The first matrix shows a neighborhood in a square while the second a neighborhood in a rhomb. We can notice from the illustration that, for example, while in the matrix to the left the v distance of the K66 unit to the WU is 1, in the matrix to the right the v distance of the K66 unit to the WU is 2.

This characteristic also allows the SOM to show the position occupied by the class within the matrix in relation to the position occupied by the other classes. This type of topological mapping, able to organize the classes through spatial relations, has been given the name of Feature Mapping.

4.9 *Topology of the neighborhood*

The neighborhood of a *WU* is defined by the degree of physical proximity (v) existing between the *WU* and the other K units. Each unit of Kohonen's layer occupies a position on the matrix of the co-ordinates (r, c) for which the neighborhood is indexed with a scalar degree from 1 to the maximum line and column dimension.

$$v_i = \pm r \quad OR \quad v_i = \pm c$$

where max *i* = max r OR max *c*

Function *h(v)* regulates the size of the neighborhood and the extent of the corrections which need to be made on the codebooks of the units close to the *WU*. With the passing of time (cycles during which all the training set models are viewed) the neighborhood is reduced until it disappears; in this case the only unit to which the codebook is corrected is the *WU*. Since the codebooks are set during the initialization stage with random values within the layering range, the proximity of the *WU* at the beginning of the learning stage is regulated with a maximum size in order to allow all the codebooks to be modified and put closer to the Input vectors. The reduced proximity with wide matrices can determine the fact that some areas of the K matrix remain isolated because the codebooks are too different from the Input vectors. Function *h(v)* must also allow the extent of the correction to be bigger for the units close to the WU, and therefore to decrease when v is larger. The Gaussian function has been shown to meet these needs remarkably well:

$$h(v) = e^{-\frac{v^2}{\sigma}} \tag{54}$$

h (v): = EXP(-(SQR (v) / σ))

where *d* is the physical proximity of the unit to the *Wu* and σ is a parameter which linearly decreases by a Δ as time increases, thereby modifying the width of the curve (bell), thus the extent of the neighborhood. Figures 11 and 12 show examples of Neighborhood Space topologies:

4.10 *Correction of the codebook*

The rate of correction a codebook undergoes is determined by various factors:

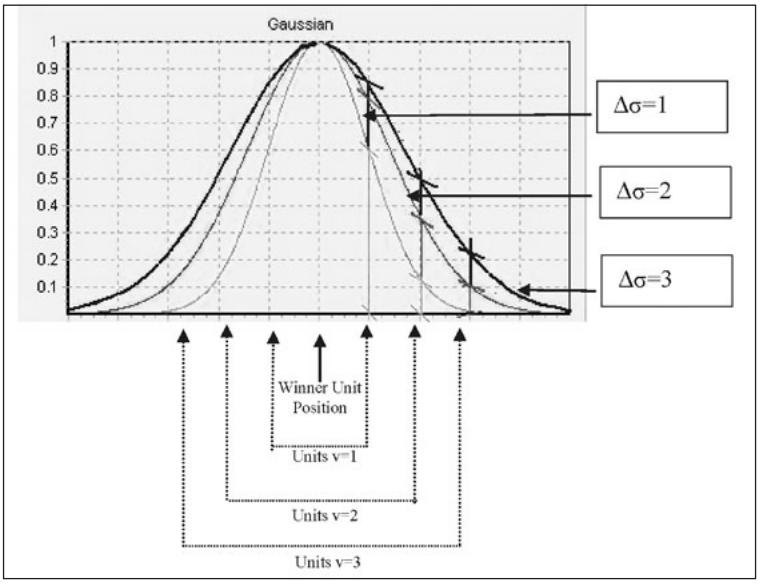

Fig. 13 – The illustration shows how, when the parameter $\Delta\sigma(1, 2, 3)$ changes – parameter that determines the correction curve of the neighborhood function – the number of units that are part of the neighborhood and the extent of the correction (v1, v2, v3) made on the weights also change.

1) Difference (d) existing between the vector codebook and the Input vector;
2) Physical distance to the WU (v);
3) Function of the neighborhood h (v) which determines a $\Delta\ \sigma$;
4) Function of weight layering in relation to the period of life of the network which determines a $\Delta\ \alpha$.

In a SOM the codebooks are moved closer to the Input vector, therefore for each generic *codebook W* the distance existing between the corresponding weights *wij* and the variables *xi* of the generic Input vector *X* is calculated. On the basis of the function h (v) of the neighborhood, the $\Delta\ \sigma$ is therefore calculated in relation to the value of the parameter s and the proximity (v) of the unit *K* to the *WU*.$\Delta\ \sigma$ is the measure which assumes y in the function *h(v)* when *x= v*. In the case in which function h(v) is the *Gaussian curve*, then the $\Delta\ \sigma$ will be calculated in the following way (Fig. 13):

$$\Delta\sigma = e^{-\frac{v^2}{\sigma}} \tag{55}$$

The $\Delta\sigma$ is calculated as a factor of a linear function decreasing in relation to the time the network is alive. Therefore, the function of correction of the codebooks is as follows:

$$f(w) = \alpha \cdot e^{-\frac{v^2}{\sigma}} \sqrt{\sum_{i=1}^{N} (x_i - w_{i,j)}{}^2} \tag{56}$$

$$w_{i,j} = w_{i,j} + \alpha(x_i - w_{i,j} \tag{57}$$

5. A Neural Networks Theory

ANNs still need a general theory able:

1) to explain which is the place that ANNs have in the general framework of the natural sciences, and
2) to explain which are the basic and atomic features we need to assemble in order to build new Neural Networks.

We can try to give a first and an approximate answer to these questions. ANNs belong to the field of Artificial Sciences. Artificial Sciences are a new and a special branch of Natural Sciences. Artificial Sciences are the new computerized laboratories through which researchers try to simulate natural processes, in order to explain their complex and hidden laws. More specifically, ANNs are a special type of computerized laboratories able to reproduce adaptive natural processes. Under this respect, data represent a statistical sample of the natural process we intend to understand. From historical point of view ANNs have shown three different targets:

1) To understand deeply the human brain work by means of its artificial simulation. In these researches, the comprehension of human brain physiology remains the main goal and ANNs are important simulation tools;
2) To develop new computation algorithms, inspired to human brain architecture, able to processes information, more effective and fast way. In this field the main goal is to define new algorithms, and the human brain structure represents only a milestone for emulation activities.
3) To understand in every natural process how the transformation works from individual behaviors to collective behaviors. In other words, how the transition works in nature from the simple and local processes to the global and complex processes. In this case the main target of the scientist is the discovery of natural laws and the ANNs represent a set of new algorithms with which we can implement the correct experiments to control our hypothesis. In this case ANN algorithms represent a powerful experimental framework for science.

These three different fields, and especially the last one, need a general theory able to explain how to build these algorithms and/or tools in a math-

ematical and physical correct way. For these reasons we are thinking to a general theory of Artificial Adaptive System.

But, it is also important to elaborate a generative model able to explain how to build new ANNs, starting from the reasoned assembly of ANNs' atomic components. For this reason we propose a bottom-up theoretical process, composed of three steps (a basic layer, a central layer, and a complex layer) and two components for each step (semantics and syntax):

1. Basic layer:
I. Semantics: Node
II. Syntax: Weighted Connection.

2. Central layer:
I. Semantics: Nodes & Connections =Networks
II. Syntax: Nodes & Connections updating under specific Learning Laws and Constraints.

3. Complex layer:
I. Semantics: Organism = Networks assembly
II. Syntax: Networks interaction under specific Signal Flow rules.

Each component of this theoretical framework needs to be analyzed in details, as we tried to do in a provisory way in a previous paper (Buscema 1998); for example: the morphology of the Node and the typology of the Connections. But this could be the main goal of a next analytical and experimental research.

Paolo Massimo Buscema
Semeion Research Center
Center for Computational and Mathematical Biology
University of Colorado at Denver

REFERENCES

Buscema P.M. 1998, *Artificial Neural Networks and Complex Social Systems. Theory,* «Substance Use and Misuse (SUM)», 33/1, 17-199.

Buscema P.M. 2007a, *A Novel Adapting Mapping Method for Emergent Properties Discovery in Data Bases: Experience in Medical Field*, in *Institute of Electrical and Electronics Engineers International Conference on Systems, Man and Cybernetics (SMC 2007) (Montreal, Canada 2007)*, 7-10.

Buscema P.M. 2007b, *Squashing Theory and Contractive Map Network*, Semeion Technical Paper # 32, Rome.

Buscema P.M., Benzi R. 2011, *Quakes Prediction Using a Highly Non Linear System and a Minimal Dataset,* in Buscema, Ruggieri 2011, 41-66.

Buscema P.M., Helgason C., Grossi E. 2008, *Auto-Contractive Maps, H Function and Maximally Regular Graph: Theory and Applications, Special Session on Artificial Adaptive Systems, Medicine: Applications in the Real World* (*New York 2008*), North American Fuzzy Information Processing Society 2008 (Institute of Electrical and Electronics Engineers).

Buscema P.M., Ruggieri M. (eds.) 2011, *Advanced Networks, Algorithms and Modeling for Earthquake Prediction,* River Publisher Series in Information Science and Technology, Aalborg, Danmark, River Publisher.

Buscema P.M., Sacco P.L. 2010, *Auto-contractive Maps, the H Function, and the Maximally Regular Graph (MRG): A New Methodology for Data Mining,* in Capecchi *et al.* 2010, 227-310.

Buscema P.M. *et al.* 2008, *Auto-Contractive Maps, An Artificial Adaptive System for Data Mining. An Application to Alzheimer Disease*, «Current Alzheimer Research», 5, 481-498.

Buscema P.M., Massini G., Newman F., Grossi E., Tasle W. 2010, *Application of Adaptive Systems Methodology to Radiotherapy*, in *Annual Meeting of the North American Fuzzy Information Processing Society, Fuzzy Information Processing Society NAFIPS (Canada 2010)*, Institute of Electrical and Electronics Engineers, Toronto, 1-8.

Capecchi V., Buscema P.M., Contucci P., D'Amore B. (eds.) 2010, *Applications of Mathematics in Models, Artificial Neural Networks and Arts*, New York-Berlin, Springer.

Chauvin Y., Rumelhart D.E. 1995, *Backpropagation: Theory, Architectures, and Applications*, Lawrence Erlbaum Associates, New Jersey, Inc. Publishers 365 Brodway-Hillsdale.

Grossi E. *et al.* 2011, *The Interaction Between Culture, Health and Psychological Well-Being: Data Mining from the Italian Culture and Well-Being Project*, J Happiness Studies, Springer.

Kohonen T. 1982, *Self-organized formation of topologically correct feature maps*, «Biological Cybernetics» 43, 59-69, reprinted from Anderson J.A., Rosenfeld E. (eds.), *Neurocomputing Foundations of Research*, Cambridge, The MIT Press.

Kohonen T. 1984, *Self-Organization and Associative Memories*, Vol. 8, Springer Series in Information Sciences, Berlin, Springer-Verlag.

Kohonen T. 1990, *The Self-Organizing Map*, Proceedings Institute of Electrical and Electronics Engineers, 78, 1464-1480.

Kohonen T. 1995, *Self-Organizing Maps*, Berlin, Springer-Verlag.

Le Cun Y., Kanter L., Solla S.A. 1991, *Second Order Properties of Error Surface Learning Time and Generalization*, Advances in Neural Information Processing Systems, Vol. 3, 918-924, San Mateo, CA, Morgan Kauffmann.

Licastro F. *et al.* 2010, *Multi Factorial Interactions in the Pathogenesis Pathway of Alzheimer Disease: a New Risk Charts for Prevention of Dementia*, «Immunity & Ageing», 7, Suppl. 1, S4.

Minsky M., Papert S. 1988, *Perceptrons,* Cambridge Ma., The MIT Press.

Rumelhart D.E., Hinton G.E., Williams R.J. 1986, *Learning Internal Representations by Error Propagation*, in Rumelhart., McClelland 1986, 318-262.

Rumelhart D.E., McClelland J.L. (eds.) 1986, *Parallel Distributed Processing, Vol.1 Foundations, Explorations in the Microstructure of Cognition,* Cambridge Ma., The MIT Press.

Werbos P. 1974, *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences,* PhD Thesis, Harvard, Harvard University.

ABSTRACT

This paper describes the philosophy of Artificial Adaptive Systems and compares it with natural language, revealing some striking parallels. Artificial sciences create models of reality, but their ability to approximate the "real world" determines their effectiveness and usefulness. This paper provides a clear understanding of the expectations created by the use of this technology, an evaluation of the complexities involved, and expresses the necessity of continuing with an open mind to unexpected and still unknown potentials. Supervised and unsupervised networks are described here.