ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/13731614

Back Propagation Neural Networks

Article *in* Substance Use & Misuse · February 1998

DOI: 10.3109/10826089809115863 · Source: PubMed

CITATIONS
117

READS
14,793

1 author:

Massimo Buscema
University of Colorado
287 PUBLICATIONS **3,505** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Semantics of Physical Space View project

Machine learning system and medical imaging View project

All content following this page was uploaded by Massimo Buscema on 09 May 2015.

The user has requested enhancement of the downloaded file.

Substance Use & Misuse, 33(2), 233–270, 1998

Back Propagation Neural Networks

Massimo Buscema, Dr.

Semeion Research Center of Sciences of Communication, Viale di Val Fiorita 88, 00144 Rome, Italy. Telephone: ++39-6-5915074. FAX: ++39-6-5920261. E-mail: semeion@ats.it. Internet: http://www.ats.it/logsis/semeion

INTRODUCTION

Back Propagation (BP) refers to a broad family of Artificial Neural Networks (ANN), whose architecture consists of different interconnected layers.

The BP ANNs represents a kind of 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.

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's 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 a 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 explicitate the *implicit knowledge* that these ANNs acquire in the training process.

A second theoretical and operative difficulty that BP poses concerns the *minimum number* of Hidden units that are necessary for these ANNs in order

Copyright © Semeion 1998

to compute a function. In fact, it is known that if the function isn't linear, at least a layer of Hidden units will be necessary. But, at the moment, stating exactly the minimum number of Hidden units needed to compute a nonlinear 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 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 isn't guaranteed; during the training process the BP can become trapped in a local's minima. This is because of the relation between the morphology complexity of the hyperparaboloidal 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 overfitting, 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 (Recurrent Networks). In this section we are going to examine only Feed Forward BP ANN, an understanding of which is essential prior to the study of Feedback BP.

STANDARD BACK PROPAGATION

Theoretical Principles

A system functioning as *Feed Forward Back Propagation* (from now on, BP), is theoretically based on the following principles:

a. The system creates the *relations* between its units, through a series of trials, with respect to multiple tasks.
b. After having learned the type of relations that are appropriate among its units, the system is able to exhibit, through its structure, the type of internal representation that it has stabilized for the various tasks it has had to learn in order to carry out the multiplicity of tasks.

c. The system can "easily" learn other tasks which are similar to the ones it has already learned, and then, to operate "generalizations".
d. The relations, which have become stabilized among the system's units during the learning of the several tasks, are the only *memory* of the system itself.
e. Many of the system's units are of a *discriminant* or subconceptual type. They can be of three logical types:
   - *Input units*: the sensors through which the system receives the surrounding area is stimulation;
   - *Output units*: the units through which the system expresses its behaviours as an *interpretation* of the received Inputs;
   - *Hidden units*: they are the internal units of the network's system. They receive the Input either from the other Hidden units or from the Input units. Their behaviour works as Output for other Hidden units or for the Output units. They are the units that provide the internal representation, through which the system interprets the Input which it receives, with respect to the Output that it will produce.
f. The relations between the system units are *uni-oriented*; that is, unit A, by activating itself in a certain way, activates unit B with respect to the *approximate* in which A activated itself and to the *strength* of their connection; but not vice versa.

This means that in BP, relations have the shape of *fuzzy rules*; for example:

if A activates in *X* way {*A*'s activation value}

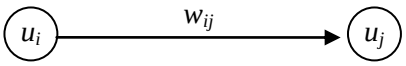
then

B will activate in *X* • *W* way {where *W* is the strength of the connection between *A* and *B*}

Since *A*'s activation in an *X* way is also the outcome of similar rule, unless *A* is an Input unit, then we can say that a BP is a continuously modifying *cascade of Fuzzy Rules*.

In short, the *power of oriented connection* between one unit and another is the fuzzy rule by which one goes from the first unit to the second one.

This strength of connection among the BP units is called *weight* and it is indicated by w_{ij}, where *i* is the starting unit's identifier and *j* is the outcome unit's identifier:



The weights among the units of a BP system are continuously modified in relation to the task that the BP has to carry out. In this sense, it is legitimate to say that the BP continuously adjust their rules to the experiences which they carry out.

This modification of the weights continues until the BP individuates the weights that will allow it to handle what it has learned up to then, in the most appropriate way.

Functioning and Learning

Given these premises, it is better to explain thoroughly the functioning of a BP.

Let's imagine a very simple BP, where we have: 2 Input units (I_1, I_2), 2 Hidden units (H_1, H_2) and 2 Ouput units (O_1, O_2). Then, it concerns a BP made up of *3 layers* (Figure 1).

Furthermore, each level of layer is connected, through weights, to all units of the following layer. The Input units are connected, through the weights $w_{I,H}$, to the Hidden units; practically, $w_{1,1}$ / $w_{1,2}$ / $w_{2,1}$ / $w_{2,2}$. Otherwise, the Hidden units are connected, through the weights $w_{H,O}$, to the Output units; practically, $w_{3,5}$ / $w_{3,6}$ / $w_{4,5}$ / $w_{4,6}$.

In order to function, a BP needs to follow several steps; the first ones follow *activation conditions*:

1. it is necessary that the BP, for at least a certain length of time, be subjected to a certain type of Input;
2. it is necessary to imagine that the Output units tend towards at least a certain type of objective called the Target, for the entire time that the BP is subjected to a certain type of Input;
3. it is necessary that the BP shows a value, even a random one at the beginning among all its unit connections that are its weights.

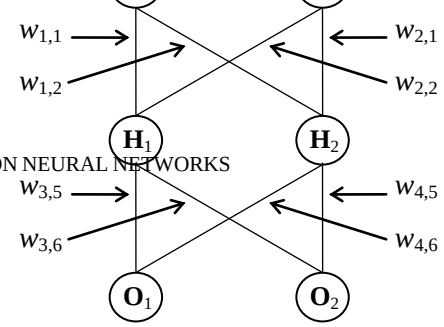


Figure 1.

The activation connections, in short, suggest that the BP must present in the initial phase:

a. at least one Input;
b. at least one Target to learn with respect to that Input;
c. random weights among its units.

We emphasize that the BP is necessary in activation prequalifications. We will express both the values of the Input units and those of the Target units with numbers ranging between 0 and 1. We will use a similar criterion in order to stabilize the random weights among the different units (Figure 2).

For example we have planned a BP system with the following task: "giving an Input of strength 1,1 determines with which *weights* the system can answer to that Input with an Output of strength 0,1".

In order *to function*, and to resolve this task, it is necessary to explain the *functioning conditions* of a BP:

1. An algorithm able to calculate the activation value of each unit, except the Input layer ones, according to the *activation value* of the units connected to it and according to the *strength* of *connection* through which these units are connected to it. We call this algorithm *Forward Algorithm*.
2. An algorithm able to gradually correct the *weights* among the different units, according to the difference between the Output produced by the forward algorithm and the desired Target. We call this algorithm *Back Propagation*.

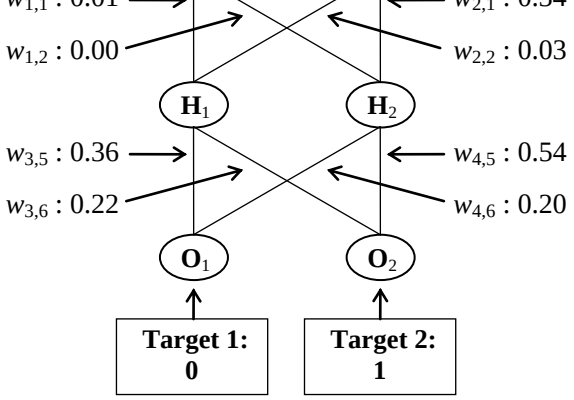


Figure 2.

These functioning conditions, presume that the BP carries out several trials in order to achieve an Output that is more similar to the desired Target. At each trial, the BP corrects its weights in order to bring the following trial as close as possible to the aim imposed from the outside.

We call a *cycle* a pair made up of a *Forward Algorithm* and the consequent *Back Propagation Algorithm*. We call an *epoch* the cycle number needed, so that ANN will have experienced, at least one time, all couples of Input and Target in order to understand them.

The *epochs* will be the lifetime of the BP. After a certain number of epochs, during which the BP has been subjected to the same Input and oriented towards the same Target, the BP is expected to have *selected* the most adequate *weights* for this aim. It is also expected that the value of these last weights and of the Hidden units will provide a good internal representation, at a subconceptual level, of the task that the BP has learned to carry out. This occurs if the *forward* and the *correction* algorithms are corrected.

Let's try then to define the *Forward Algorithm*.

The basic rule in order to calculate the activation value of a unit in respect to other units which are connected to it, with a strength w_{ji}, is a function of the weighted sum of the Inputs:

$$(1) \qquad u_j = f\left(\sum_i u_i \cdot w_{ji}\right) = f(Net_j) = \frac{1}{1+e^{-Net_j}}$$

where: Net_j = Net Input to the j level unit

One must add to this equation the *threshold* of the unit, described as the *inclination* of the unit to activate or inhibit itself.

This means that:

(1a) $$u_j = f(Net_j) = \frac{1}{1+e^{-\left(\sum(i) w_{ji} \cdot u_i + \theta_j\right)}}$$

where θ_j is the *Bias* of unit u_j; this is the degree of *sensitivity*, with which the unit u_j answers to the perturbation it receives by the Net Input. The Bias is the opposite of the threshold and it behaves as a weight generated by a fixed Input of unitary value.

Equation (1a) represents the forward algorithm of BPs, assuming as default, the sigmoidal function.

It is necessary to calculate the dynamic of the back propagation or correction algorithm.

The mathematical basis of this algorithm was already individuated by Rosemblatt in 1956, through the so-called *Delta Rule*, a procedure that allowed to correct for excess or defect of the weights between the network units, on the basis of the difference between the actual Output and the desired Target.

Nevertheless, the Delta Rule allows correcting only those weights that connect the Output units with those of the just underlying layer. It doesn't allow one to know in a 3-layer network, how at each cycle, the weights that connect the Input units with the Hidden units, should be modified.

Let's examine in detail the Delta Rule.

The coefficient of error in this procedure is calculated by considering the difference between the actual Output and the desired one (the Target one) and relating this difference to the derivative between the activation state of the actual Output and the Net Input of that Output.

Therefore, if $\frac{\partial u_j}{\partial Net_j} = u_j \cdot (1 - u_j)$, then Δout_j (this is the error coefficient) will be:

(2) $$\Delta out_j = (t_j - u_j) \cdot u_j \cdot (1 - u_j)$$

where: t_j = desired Output (Target); u_j = actual Output; $u_j \bullet (1–u_j)$ = derivative between actual Output and Net Input of unit u_j.

This is based on the fact that:

(3) $$\Delta w_{ji} = -\frac{\partial E_p}{\partial w_{ji}};$$

a) $$E_p = \tfrac{1}{2}\sum_k (t_{pk} - u_{pk})^2 = \tfrac{1}{2}\sum_k (t_{pk} - f_k(Net_{pk}))^2 =$$

$$= \tfrac{1}{2}\sum_k \left(t_{pk} - f_k\left(\sum_j w_{kj} \cdot u_{pj} + \theta_k\right)\right)$$

where: E = Error; p = Model; t_k = Target; u_k = Output.

And then:

b) $$\frac{\partial (Net_{pk})}{\partial w_{kj}} = \left(\frac{\partial}{\partial w_{kj}} \cdot \sum_j w_{kj} \cdot u_{pj} + \theta_k\right) = u_{pj} \ ;$$

c) $$-\frac{\partial E_p}{\partial w_{kj}} = (t_{pk} - u_{pk}) \cdot f_k^{'}(Net_{pk}) \cdot u_{pj}$$

At this point, one can say that the quantity of value to be added or subtracted from weight w_{ji} will be decided by value Δout_j, with respect to the activation state of unit u_i, namely, the activation with which u_j is connected to weight w_{ji} and in relation to the coefficient r. This is the correction rate that one wants to adopt (when r = 1, then the value of the adding or subtracting from weight w_{ji} is the one calculated by the whole procedure).

(4) $$\Delta w_{ji} = r \cdot \Delta out_j \cdot u_j$$

The value Δw_{ji} can be both negative and positive. It represents the "quantum" to be added or subtracted from the previous value of weight w_{ji}.

Then:

(5) $$w_{ji(n+1)} = w_{ji(n)} + \Delta w_{ji}$$

Nevertheless, equation (2) presupposes that each arriving unit of a weight has an actual value which is comparable with an ideal value, towards which it should tend (Target). This presupposition, however, is valid only for the weights that connect a unit layer with the layer of the Output units.

The correction procedure discussed till now, related only to BP provided with 2 layers (in reality, 1 layer if we consider that the Input unit can't be considered as 1 layer of the network).

The Delta Rule, then, represented by equation (2), allows one to carry out the weight's correction only for very limited networks. For *multilayer* BP, this is, with 1 or more layers of Hidden units, the Delta Rule, and as it is, is insufficient; for example:

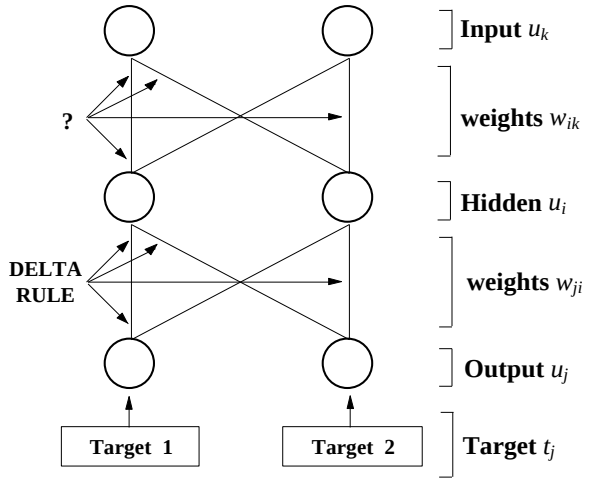


Figure 3.

In Figure 3 it is evident that, while the correction of weights w_{ji} is possible through the Delta Rule, because the value that the units u_j should assume (t_j) is known, the correction of weights w_{ik} is not possible. This is so because a value of ideal reference doesn't exist for the units u_i.

In fact, the value that they are going to assume is one of the *results* of BP's learning work and therefore it can't be constrained.

Rumelhart and others solve this problem through a *generalization* of the traditional Delta Rule (Rumelhart, 1986a).

The generalization of the Delta Rule consists in modifying equation (2), in those cases in which the weight to modify isn't connected to an Output unit.

Therefore, instead of computing the difference between the actual Output and the desired one, a report summation will be computed between the error coefficient Δout_j, previously calculated, and the weights which that coefficient was referring to:

(3a) $\Delta hidden_i = u_i \cdot (1-u_i) \cdot \sum \Delta out_j \cdot w_{ji}$

and therefore:

(4a) $\Delta w_{ik} = r \cdot \Delta hidden_i \cdot u_k$;

(5a) $w_{ik(n+1)} = w_{ik(n)} + \Delta w_{ik}$.

In fact, starting from equation (a), we have that:

(d) $$\frac{\partial E_p}{\partial w_{kj}} = \frac{1}{2} \sum_k \frac{\partial}{\partial w_{kj}} \cdot (t_{pk} - u_{pk}) =$$

$$= -\sum_k (t_{pk} - u_{pk}) \cdot f_k(Net_{pk}) \cdot w_{kj} \cdot f_j(Net_{pj}) \cdot u_{pi}$$

where w_{kj} = Hidden-Outputs weights.

Through the generalized Delta Rule it is possible to create a back propagation algorithm, capable of correcting the weights of any BP's layer at every cycle.

We can now synthesize the 2 algorithms through which the BP would be able to work:

a. Forward Algorithm

1) $Net_j = \sum_i u_i \cdot w_{ji} + \theta_j$

2) $u_j = f(Net_j)$

b. Back Propagation Algorithm

b1. *Correction calculation of the weights connected to the Output*:

1) $\Delta out_j = (t_j - u_j) \cdot f'(u_j)$

2) $\Delta w_{ji} = r \cdot \Delta out_j \cdot u_i$

b2. *Correction calculation of the weights not connected to the Output*:

1) $\Delta hidden_i = f'(u_i) \cdot \sum_j \Delta out_j \cdot w_{ji}$

2) $\Delta w_{ik} = r \cdot \Delta hidden_i \cdot u_k$

b3. *Fulfillment of the corrections on the weights*:

1) $w_{ji(n+1)} = w_{ji(n)} + \Delta w_{ji}$

2) $w_{ik(n+1)} = w_{ik(n)} + \Delta w_{ik}$

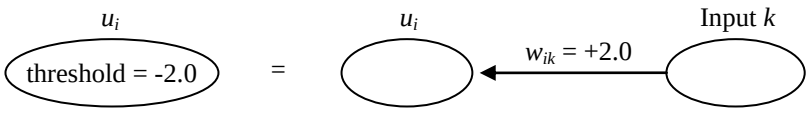
At this point, both the *minimum conditions of activation* and those of *functioning*, or learning for BPs, have been explained.

The Self-programming Bias

The Bias, both conceptually and mathematically, is the unit's *threshold*. Nevertheless, from an arithmetical point of view, it is expressed through a value of sign opposite to that of a threshold. For example: if unit u_i has a threshold of -2.0, then its Bias will be +2.0.

This means in both cases that unit u_i will be active until it won't receive a Net Input less than -2.0.

Conceptually, the Bias is the inclination of a unit presented as a *weight* that arrives from an Input, fixed on value 1, to that unit:



this is the reason for which the Net Input is calculated by adding the Bias value to the sum of the products between the units and the weights afferent to the reference unit at every unit:

$$(6) \qquad u_i = f\left(Net_i\right) = f\left(\sum_{j=1}^{N} u_j \cdot w_{ij} + Bias_i\right)$$

where $Bias_i$ = - $Threshold_i$

Considered the Bias's structural and functional nature, its dynamic for each network unit can be considered similar to that of any weight. Therefore, the weights' matrix is liable to the normal learning algorithm to which it is subjected.

This means that every BP can be provided with *dynamic threshold's* units. Each unit will dynamically learn its threshold, in relation to the type of experiences that the whole ANN is carrying out.

This means that if the updating of the weights through the Back Propagation is given by the following equations:

$$(7) \qquad \Delta out_i = (t_i - u_i) \cdot f'(u_i)$$

$$(8) \qquad w_{ij(n+1)} = w_{ij(n)} + \Delta out_i \cdot u_j \cdot Rate$$

for the Output units; and

$$(9) \qquad \Delta hidden_j = f'(u_j) \cdot \sum_{i=1}^{N} \Delta out_i \cdot w_{ij}$$

$$(10) \qquad w_{jk(n+1)} = w_{jk(n)} + \Delta hidden_j \cdot u_k \cdot Rate$$

for Hidden units, then equations (11) and (12) will determine the Bias updating of the Output layer and of any Hidden layer:

$$(11) \qquad Bias_{i(n+1)} = Bias_{i(n)} + \Delta out_i \cdot Rate$$

$$(12) \qquad Bias_{j(n+1)} = Bias_{j(n)} + \Delta hidden_j \cdot Rate$$

It is correct to consider the Bias of a unit as being a *dynamic threshold*, subject to learning which from an algebraic point of view can be formulated as a dynamic weight generated from a fixed Input of value 1 towards the unit.

Then, the Bias represents the *historical and individual sensitivity* of each unit of ANN to the experiences of the whole network.

The self-programming Bias has considerably increased the learning rapidity of BP ANNs and has, of course, decreased the possibility of entrapment of the network to local minima.

The Momentum

Experience has shown that the more that the learning coefficients of equations (7) and (8) are reduced, the less the probability that the network entraps itself in local minima.

At the same time, the more that the learning coefficient is small, the slower is the learning.

An attempt has been made to resolve this dilemma through the introduction of a new parameter, the *Momentum* (McClelland, 1988:135).

The *Momentum* is a parameter through which the network reinforces the change of each of its connections in the descent's general direction of the paraboloidal, which has already emerged during its previous process for updating. The reason for this is the eventual deleting contingent upon oscillations produced by the steepest-descent algorithm.

With the introduction of the *Momentum*, equation (7) becomes:

(7a) $\Delta out_{i(n)} = (t_i - u_i) \cdot f'(u_i)$

(7b) $Momentum_{ij(n)} = \Delta w_{ij(n-1)} \cdot k \qquad 0 < k \leq 1$

(7c) $\Delta w_{ij(n)} = Momentum_{ij(n)} + \Delta out_{i(n)} \cdot u_j \cdot Rate$

where $Rate_i$ is the Learning Rate of the *i*-th level unit.

The introduction of the *Momentum* also allows the ANN learning to speed up by utilizing rather small learning coefficients.

The experience of many authors recommends a value of $k = 0.9$, with a learning coefficient, $Rate = 0.6$.

Nevertheless, the major conclusion which is emerging from this research seems to offer other suggestions:

a. it is not reasonable to suggest values for the *Momentum* without knowing the problem's typology which the ANN must resolve;
b. it is good to suspect every equation in this scientific field from which the researcher expects decisive constants: whether they are useful only for certain experiments or hide further relations that must be explicitated.

Inserting arbitrary constants in these models is equivalent in a sense to adding *symbolic rules* to *subsymbolic procedures*.

Moreover, the Momentum does not eliminate the theoretical possibility that the ANN occurs in the local minima.

The Transfer Equations

Until now, the transfer function implicitly used in the previous equations has been the *sigmoidal* one.

Nevertheless, equation (6) wasn't specific on this matter:

$$(6) \qquad u_i = f\left(Net_i\right) = f\left(\sum_{j=1}^{N} u_j \cdot w_{ij} + Bias_i\right)$$

If we mean that *Net* is the Net Input to a unit, and that *f* ( ) is the equation of the Sigmoid, equation (6) will be rewritten in the following way:

$$(13) \qquad u_i = \frac{1}{1+e^{-Net_i}} \qquad 0 \le u_i \le 1$$

e = 2.718281828459

Through this equation, the activation value of u_i varies between 0 and 1 according to a semilinear functioning which has its flex point as 0.5.

In fact: if $Net_i = 0$, then $u_i = 0.5$.

Therefore, its derivative will be: $f'(Net_i) = u_i \cdot (1-u_i)$.

This function is the most diffused in Back Propagation ANNs and some originators have tried to make it more complete, through the introduction of other parameters; for example:

$$(13a) \qquad u_i = \left[1+e^{\frac{-Net_i}{T}}\right]^{-1}$$

where *T* is a parameter called temperature (see below).

For $T = 0$, the function is reduced to a *degree* exit 1 or 0, while with the growing of *T*, the slope of the sigmoidal (for a closer examination, see below) increases.

A second variation of (13) is the following:

$$(13b) \qquad u_i = \frac{coef}{coef + e^{-Net_i}}$$

where for $coef > 1$ the unit sensitivity to the Net Input value increases, while for $coef < 1$ this sensitivity decreases.

This equation can be useful in order to avoid the fact that in some problems the Hidden units are all overloaded towards the value 0.0 or towards value 1.0 without being able to codify the differences among the different Input models (for a closer examination, see Buscema, 1993:38).

A second transfer equation which is often used is that of the *Hyperbolic Tangent*:

$$(14) \qquad u_i = \frac{e^{Net_i} - e^{-Net_i}}{e^{Net_i} + e^{-Net_i}} \qquad -1 \le u_i \le +1$$

In this case, the values of u_i vary between -1 and +1; therefore, the derivative of this function, at the moment of the weights' correction, will have to be: $(1.0 + u_i) \cdot (1.0 - u_i)$ to replace the derivative of the sigmoidal function $u_i \cdot (1 - u_i)$.

The equation of the Hyperbolic Tangent is less soft than the sigmoidal function. This means that it can be less useful with problems which expect a fuzzy Target or Input vectors with no determinant differences.

When there is the possibility of not making fall ANN in local minima, the Hyperbolic Tangent is preferable to the Sigmoid for learning velocity about binary problems (where the Input and Target vectors are laces of 0 and 1) and when strong corrections are desired for values near to the limits of -1 and +1.

A transfer equation that presents similar advantages, but with less imprecision, is that of the *Archtangent* which has shown itself to be efficient or not a hindrance in the solution of complex problems:

$$(15) \qquad u_i = \frac{1}{2} + \frac{1}{\Pi} \cdot Arctg(Net_i)$$

In (15) u_i varies between 0 and 1 and presents itself as a sigmoid gentler than the traditional sigmoid of equation (13).

A transfer equation that requires a different treatment is called *sinusoidal*:

(16) $u_i = sin(Net_i)$

Firstly, it is important to notice that making use of this function requires modifying the derivative in the equations of weights' correction: $u_i \cdot (1 - u_i)$ becomes in the sigmoid $cos(Net_i)$ in the sine function.

Its characteristic consists in making clear a discrete class of couples (x, y) in synthesizing the continuous function $y = g(x)$ (considering an ANN with only 1 Output). Practically, it seems, it carries out a decomposition in the principal components of the function described in a discrete way from the Input-Output examples.

We noticed in many experiments that by selecting for the layer of Hidden unit a half unit with the sigmoid and a half with the sine, *two* split halves of the Input models are self-created. The first, constituted by the sigmoidal units, tends to codify the determinant differences of the details among Input models, while the second, constituted by sinusoidal units, tends to codify the similarity of the details between the different models.

This peculiarity, also noticed by other authors (NeuralWare, 1993), often guarantees a better generalization capacity to the whole ANN.

Only the functions of *linear*, *degree* (*Hard Limiter*) and *Slope transfer* remain to be discussed. Their use in BP ANN is replaceable by some functions already treated or unsatisfying.

In reality, the choice of the transfer functions is just one of the choices that determines the functioning of an ANN. Moreover, this choice should be made while considering all the other choices that must be made: type of problem, topology of ANN, Learning Rates, type of Momentum, type of Back Propagation, etc.

Therefore, it is useless to assert *a priori* which of those equations is more efficacious, except for rare cases (for example, the weakness of the linear transfer in Hidden units).

Further on, we will see that their efficacy changes according to the types of problems and to the global architecture of ANN.

On the basis of research implemented since 1985 for Back Propagation ANNs, perhaps it can be asserted that the sigmoidal function has always shown a good behaviour in very different cases and problems.

METHODOLOGICAL DEVELOPMENTS

The SoftMax

This is a function utilized in the learning process in BP Networks for the classification of problems.

In order to resolve a problem of this type, the desired Output (Target) is usually represented through the code 1 of N classes. Each class is represented by an Output unit; then, in the case of N classes, the Target will be constituted by a *vector* composed by N Output nodes, and it will have the following form:

$\underline{d}$ = (0, ..., 0, 1, 0, ..., 0)

where only the unit that corresponds to the desired class has a Target value different from 0 (that is, 1).

There are 2 problems with this formula:

1. The codified 1 of N is dependent on the values of the Output units. When one of the units has a value equal to 1, the other units have 0 value. Nevertheless, there is no assumption of dependence in the error in a trained back propagation network.
2. If there are more classes, the Network can find a reasonable solution through a mapping of the Output vector. This will give an RMS of: $1/\sqrt{k}$ (where k is equal to the class number).

This solution is a very simple to find. For example, if we use the sigmoid as transfer function, and all the weights of the Output layer are big and negative, all the Output units will have the Outputs equal to 0. If the class number is not too big, placing a very low learning rate, the network will have difficulty converging on this artificial solution.

In order to have a satisfying solution for these 2 problems, it has been recommended that the "SoftMax activation function" be used for the layer of Output units (Bridle, 1989).

The SoftMax function is a refined version of the competitive function 1 of N, and it has some convenient mathematical properties:

$$y_k = \frac{e^{I_k}}{\sum_{l=1}^{k} e^{I_l}}$$

Bridle proposed that the SoftMax function be utilized by joining it with the function of the following aim, which is derived from the relative entropy

measure between the Target and the actual Output:

$$J = -\sum_{j=i}^{k} d_j \ln(y_j)$$

Let's calculate the value that will return from the Output unit j in the following way:

$$\frac{-\partial J}{\partial I_j} = -\sum_{k=1}^{k} \frac{\partial J}{\partial y_k} \cdot \frac{\partial y_k}{\partial I_j}$$

This starts from the quotient rule:

$$\frac{\partial y_k}{\partial I_j} = y_k(\partial k_j - y_j)$$

Therefore:

$$\frac{-\partial J}{\partial J} = -\sum_{k} \left(\left(\frac{-d_j}{y_k} \right) \cdot y_k \cdot \left(\delta k_j - y_j \right) \right) = d_j - y_j \sum_{k} d_k = d_j - y_j$$

where: $\delta k_j = \begin{cases} 1 \text{ if } k = j \\ 0 \text{ if } k \neq j \end{cases}$

This means that the back propagation's standard algorithm can be used with the SoftMax function for the back propagation of the real error.

The Fast Propagation

In order to speed up the learning of Back Propagation ANNs, Tariq Samad has changed equation (8) in the following way:

(8a) $w_{ij(n+1)} = w_{ij(n)} + Rate \cdot \Delta out_i \cdot (u_j + k \cdot \Delta hidden_j)$

This means that, to the value of the rising unit u_i of the connection, the

error that it has registered in the previous cycles is added in proportion k ($k > 0$; for $k = 0$, it is the normal back propagation of equation (8)).

NeuralWare (1993) has implemented this technique by defining it "Fast Propagation" and sustaining its advantages in terms of rapidity (for comparisons, see M. Buscema, 1994, *Squashing Theory*, Armando, Rome, p. 102; while for a closer examination of this technique, see Samad 1988 and 1989).

Semeion's SelfMomentum

An attempt to resolve the problem of how these networks learn rapidity was done in a slightly different way in 1989 (M. Buscema, 1989, December: Tests at Semeion's).

The hypothesis was that it was necessary to reinforce the descending direction of the paraboloidal as a function of the error in the moment that each node was generating.

In other words, it is good to spank the child in order to remind him about the good examples when he does wrong, but it is silly to spank him every time he moves.

More formally, starting from (7a), the *SelfMomentum equation* appears in the following way:

(7a) $$\Delta out_{i(n)} = (t_i - u_i) \cdot f'(u_i)$$

(17) $$SelfMomentum_{ij} = \Delta w_{ij(n-1)} \cdot \left|\Delta out_{i(n)}\right| \cdot \frac{1}{0.5 + \left|w_{ij}\right|}$$

(18) $$\Delta w_{ij(n)} = SelfMomentum_{ij(n)} + \Delta out_{i(n)} \cdot u_j \cdot Rate$$

where: $|w_{ij}|$ = absolute value of connection w_{ij}

The *SelfMomentum* eliminates the arbitrary parameter k of the *Momentum* and allows ANN to be able to stabilize in an autonomous way the strength with which the direction of the weights' correction is reinforced.

On practical level the *SelfMomentum* equation allows all the problems to be resolved through the *Momentum*, by maintaining the coefficient of unitary learning (*Rate* = 1) and in a faster way.

Figure 4 shows the progress of the Self Momentum formulating

$\Delta out_{i(n)} = 1$, varying the $\Delta w_{ij(n-1)}$ in the interval [-0.5, 0.5] and the w_{ij} weights in the interval [-1.0, 1.0].

The Neuron's Temperature: Adaptive Neuron Model (ANM)

In 1988 Raul Tawel of the Jet Propulsion Laboratory (California Institute of Technology) proposed a *learning* model for neurons in addition to one for the weights: the ANM (Adaptive Neuron Model) model.

Tawel's thesis is founded on the biological nonsubstantiality of considering the "artificial neurons" – that is, the units of ANN – as *passive elements* in the learning process.

Tawel proposes, then, a learning model also for the *units* of an ANN: the ANM.

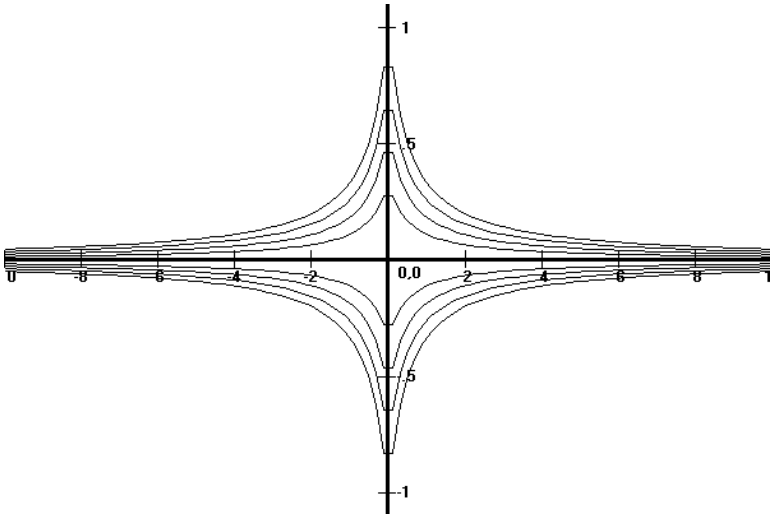


Figure 4. Progress of the SelfMomentum on the ordinate;
value of the weights in the interval [-1.0, 1.0] on the abscissa.

The ANM allows every neuron, during the learning process, to develop its own uniqueness as one of any connection (weight).

Each ANM's unit expresses this individuality through the parameter of its specific *temperature*. The individuation of a singular temperature for each ANN's unit implies that each unit's *incremental change of temperature* is

proportional to the negative of the error derivative in respect to the temperature of that unit:

$$(19) \quad \Delta T_i^{[s]} = -\hbar \frac{\partial E}{\partial T_i^{[s]}}$$

where: $\Delta T_i^{[s]}$ = increase or decrease of the Temperature of *i*-th level unit [s]; $\hbar$ = learning coefficient of the units (same concept to that of the weights' Rates); $T_i^{[s]}$ = actual Temperature of the *i*-th level unit [s]; E = Global Error (see equation (25));

from which descends:

$$(20) \quad \frac{\partial E}{\partial T_i^{[s]}} = -\Delta_i^{[s]} \cdot \frac{\partial f}{\partial T_i^{[s]}}$$

where $\Delta_i^{[s]} = \Delta out_i$ or $\Delta hidden_i$, if unit u_i is of Output or Hidden.

and finally:

$$(21) \quad \frac{\partial f}{\partial T_i^{[s]}} = -\frac{Net_i^{[s]} + \theta_i^{[s]}}{\left(T_i^{[s]}\right)^2} \cdot \frac{e^{-\left(\frac{Net_i^{[s]}+\theta_i^{[s]}}{T_i^{[s]}}\right)}}{\left(1+e^{-\left(\frac{Net_i^{[s]}+\theta_i^{[s]}}{T_i^{[s]}}\right)}\right)^2}$$

The increasing factor $\Delta T_i^{[s]}$, obtained for a certain unit, is added to the Temperature of that unit:

$$(22) \quad T_i^{[s]}(n+1) = T_i^{[s]}(n) + \Delta_i T_i^{[s]}$$

and then computed as its new temperature in the transfer equation that changes the Net Input of that unit in its activation value:

$$(23) \quad u_i^{[s]} = \left(1 + e^{-\left(\frac{Net_i^{[s]} + \theta_i^{[s]}}{T_i^{[s]}} \right)} \right)^{-1}$$

if the utilized equation is the sigmoid equation.

In this way in ANM, an ANN learns simultaneously the weights' matrix and the temperature vector of all its units.

In some simulations concerning the "XOR" problem utilizing a Back Propagation ANN, Tawel states the capacity of its model to avoid local minima and to speed up the convergence times of the network (about 1000 cycles).

In these simulations Tawel has set the learning coefficient $\hbar = 0.1$ for the temperature, and $n = 0.1$ for the weights; he has allowed a total temperature variation between +0.01 and +2.00; he has randomly initiated the units' initial temperatures in a space included between +0.9 and +1.1, while the weights have been randomized in a space included between +2.00 and -2.00.

Tawel is right, both from the epistomological and biological point of view: the units of an ANN can't behave as passive elements. Perhaps Tawel is also right, when he states that ANN's units must *participate* in the learning process, in an analogous way to how the synapsis (weights' matrix) participate in it.

Probably, Tawel is also right when he hypothesizes the units' learning algorithm by instituting a proportion between the change of a parameter, $\Delta T_i^{[s]}$ and the negative of the Global Error's derivative of ANN in respect to the parameter $T_i^{[s]}$ of that unit (equation (19)).

We doubt that the parameter $T_i^{[s]}$ influence corresponds to the unit temperature and then the parameter $T_i^{[s]}$ updated through $\Delta T_i^{[s]}$ must not, for each unit, affect the sigmoid form. And then, that equation (23) is not adequate to the problem. This inadequacy has different reasons:

a. The experimental results have shown that the ANM model obtains contradictory results according to the type of problems; and in the case in which it obtains better results, these are clearly inferior to those obtained through another conception of the temperature concept (see section on "*Freezing*").
b. It is difficult to believe that the learning of each unit is reduced in determining its sigmoid form.

c. Rather, it is more probable that the temperature of all neurons is linked to the nature of each Model that ANN experiments, and to the Error that this Model determines. It is a *reactive* and *contingent* and not a learning behaviour (see section on "*Freezing*").

Semeion's Freezing

In an attempt to resolve the problem of *convergence rapidity* and of falling in *local minima* of the BPs, a new equation that adjusted the temperature of each Model with each ANN was tested in 1989. This new technique has been called *Freezing* (M. Buscema, 1989, March: Experiments at Semeion).

Freezing bases itself on the following assumptions:

a. each Model establishes a temperature proportional to the Error that it determines in the whole ANN;
b. when the Global Error tends to zero, the temperature that each Model determines in ANN also tends to zero; therefore:

$$(24) \quad T_p = f(Error_p) = f\left(\frac{1}{2} \cdot \sum_{i=1}^{N} (t_i - u_i)^2\right)$$

$$(25) \quad T_p = f(Error_p) = 1 - \frac{1}{1 + Error_p} \qquad \text{where } p = \text{Pattern}$$

from which:

$$(26) \quad u_{pi} = \left(1 + e^{-\left(\frac{Net_{pi} + \theta_{pi}}{T_p}\right)}\right)^{-1}$$

if the transfer equation is the sigmoid.

The *Freezing* presupposes that all units have the same temperature in respect to a Model. This temperature is determined by the Error that that Model has produced.

Furthermore, the *Freezing* presupposes that the different Models in

relation to the Error that they determine can continuously increase or decrease the global temperature of ANN.

This means that the unit becomes *softer* for the Models for which the Error is higher (the temperature increases) and more *rigid* (till the Hard Limiter) for the Models for which the Error is lower (the temperature decreases till values approach zero).

This would also mean that ANN's learning coincides with the *homogenization* of ANN's temperature for all Models. The learning coincides with the capacity of ANN to be able to recall each Model with 0 and 1 limit values.

The *Freezing*, in this sense, is similar to a process of "simulated annealing" that is self-generated by ANN itself.

From an epistemological point of view the *Freezing* has, in its simplicity, many advantages:

a. Considering the Error of each Model as the Attrition produced by the application of a strength on a body, the increase of the Attrition determines the temperature's increase of ANN. In this sense the temperature creates an effect of other mechanisms' functioning and is not simply one "more" parameter "more". The attrition produced for each Model is the Noise (Error) that ANN must eliminate in order to be able to compute the Input/Output function.
b. The end of a learning in an ANN corresponds to its functional death: the ANN can recall and show how much, how and what it has learned, but this is an experimental artifice. An ANN should always be in a learning function. Through the *Freezing* in ANN, the end of the learning corresponds to its "crystallization".
c. the *Freezing* documents how each ANN tends to recall its own units through binary choices (1/0), and that this result is reachable only after the learning has happened.

The *Freezing* presents advantages also from the experimental point of view:

a. ANN converges more rapidly and with very low Global Error values;
b. ANN gets out autonomously from the local minima;
c. ANN minimizes the modification of its own weights, thus avoiding the overfitting problem, this is, the noise codification in the "real data" (see Weigend, 1991).

The Quick Propagation

The Quick Propagation algorithm has been planned by Scott Fahlman, at

Carnegie's Mellon University, with the aim to improve the convergence rate in BP networks (Fahlman, 1988).

In order to find an optimal solution in the shortest time, it is not possible to proceed in the weight's space following the gradient with very short infinitesimal steps. What is realistically asked is to proceed towards the minimum with the most possible strident steps, avoiding the rebound phenomenon on the minimum surfaces without reaching it.

Furthermore, in the hypotheses that the changes carried out on a coefficient do not have much influence on the rate $\partial E / \partial w$ pertinent to another coefficient, and that E is approximately a squared function of each coefficient's value (in the case of linear units it would be exactly a squared function), it can be considered that the knowledge of the three memorized values defines, on a Cartesian plane of coordinates w, E, a parabola that graphically represents the dependence law of E from the considered coefficient w. The learning law of the Quick Propagation consists, then, in choosing a variation of this connection coefficient so that it assumes the value corresponding to the vertex abscissa of the parabola, where E assumes the minimum value possible in that moment.

Quick Propagation resolves this by combining two traditional approaches:

a. dynamic regulation of the learning rate (globally or separately for each weight) based on the historic succession of the learning rate up to the actual value;
b. calculation of the second derivative of the error in respect to each weight.

The Quick Propagation is more of a heuristic method than a formal one of the second order, and it is vaguely based on Newton's method.

The parabola is determined by calculating for each weight the slope of the former and the actual error and the changing of the weights' values among the points for which these slopes are measured.

The correction algorithm allows to directly reach the minimum of this parabola.

Generally, the Quick Propagation characterizes itself very well in respect to the other learning techniques. It is often in competition with the Extended-Delta-Bar-Delta method and the Conjugate Gradients' method, particularly concerning problems with "clean data"; these which are not noisy.

Through the Quick Propagation algorithm, all the weights are independently updated; in particular, for each weight, $w_{ij(n)}$, are memorized:

a. the value of $\partial E / \partial w_{ij(n)}$ at the end of the previous cycle, $\partial E / \partial w_{ij(n-1)}$;

b. the value of $\partial E / \partial w_{ij(n)}$ at the end of the present cycle;

c. the value of $\Delta w_{ij(n)}$ calculated at the end of the previous cycle, $\Delta w_{ij(n-1)}$.

At the end of the present cycle the coefficient variation $\Delta w_{ij(n)}$, must be:

$$\Delta w_{ij(n)} = x_0 - \Delta w_{ij(n-1)}$$

where x_0 is the vertex abscissa of the parabola, calculated in the following way:

$$x_0 = \frac{-\frac{\partial E}{\partial w_{ij(n-1)}} \cdot \Delta w_{ij(n-1)}}{\frac{\partial E}{\partial w_{ij(n)}} - \frac{\partial E}{\partial w_{ij(n-1)}}}$$

If during the learning, a weight $w_{ij(n)}$ assumes a value lower than a predefined constant threshold, it will be posed equal to zero.

The Delta-Bar-Delta and the Extended Delta-Bar-Delta

The Delta-Bar-Delta (DBD) algorithm, planned by Jacobs (Jacobs, 1988), is a heuristic trial in order to improve the convergence velocity of the weights of a multilayer network.

Jacobs suggests the application of a heuristic law which takes into consideration the variation of the error surface in respect to the generic weight. In particular, each network connection must have a specific learning coefficient.

Jacobs's idea is that the same correction rate can't be adequate for all weights. Furthermore, the learning coefficients should vary in time (as cycle numbers intended).

In DBD, as in the standard Delta Rule, the components of the error's gradient in respect to the weight w_{ij}, calculated in the following way:

$$\Delta w_{ij} = -\frac{\partial E_{(n)}}{\partial w_{ij(n)}}$$

determine the contribution to the error due to the particular weight w_{ij} of the network. Furthermore, if in the standard Delta Rule the Rate is constant for

the weights' updating, in the DBD a variable Rate $\alpha_{(n)}$ is assigned to each network's weight. Then, the updating of every weight will be given by the following equation:

$$w_{ij(n+1)} = w_{ij(n)} + \alpha_{ij(n)} \cdot \Delta o_{(n)}$$

where: $\Delta o_{(n)} = \Delta out_{i(n)} \cdot u_j$ (where u_j is the starting unit of connection w_{ij}).

Jacobs, in order to implement this heuristic law for the increase or decrease of the learning rates of each connection, has utilized a weighed mean of the gradient's $\Delta o_{(n)}$ components, which is calculated as:

$$\overline{\Delta o}_{(n)} = (1-\varphi) \cdot \Delta o_{(n)} + \varphi \cdot \Delta o_{(n-1)} \qquad 0 \le \varphi \le 1$$

where φ is a coefficient for weighing the linear combination of the present term Δo and that of the previous cycle.

The rate of the variable learning $\alpha_{ij(n)}$ obeys the following law:

$$\alpha_{ij(n+1)} = \alpha_{ij(n)} + \Delta\alpha_{ij(n)}$$

In order to determine $\Delta\alpha_{ij(n)}$, we have that: if the component of the present gradient and the exponential mean of the previous components of the gradient have the same sign, then the learning rate associated with that weight will be increased by a constant value *k*; vice versa, if the sign is different, the learning rate will be decreased by a quantity proportional to its actual value.

Therefore:

$$\textit{if} \quad \left(\overline{\Delta o}_{(n-1)} \cdot \Delta o_{(n)}\right) > 0 \qquad \textit{then} \quad \Delta\alpha_{ij(n)} = k$$

$$\textit{if} \quad \left(\overline{\Delta o}_{(n-1)} \cdot \Delta o_{(n)}\right) < 0 \qquad \textit{then} \quad \Delta\alpha_{ij(n)} = -\psi\alpha_{ij(n)}$$

$$\textit{otherwise} \qquad\qquad \Delta\alpha_{ij(n)} = 0$$

It has to be noted that DBD increases the learning rates linearly, but decreases them geometrically. In order to obtain a good convergence, *k* must be less than ψ. It is advisable to put $k = 0.001$ and $\psi = 0.005$.

In 1990 Minai and Williams (Minai, 1990), have planned the *Extended-Delta-Bar-Delta* (EDBD) starting from some considerations about Jacobs's DBD:

a. the standard DBD does not utilize the Momentum;
b. small and linear constant (*k*) increases can modify the learning rate, determining limits in the weights' space which make the network convergence difficult;
c. the geometric decrease sometimes isn't sufficiently fast enough in order to avoid limits in the weights' space.

A variable coefficient, $\mu_{(n)}$, of the Momentum's modulation has been introduced, in the EDBD, for each network's connection. In the Delta Rule, classical with the Momentum, the coefficients *Rate* and *k* are constant, while in EDBD they vary in time:

$$\Delta w_{ij(n+1)} = \alpha_{ij(n)} \cdot \Delta o_{(n)} + \mu_{ij(n)} \cdot \Delta w_{ij(n)}$$

and then:

$$w_{ij(n+1)} = w_{ij(n)} + \Delta w_{ij(n+1)}$$

In EDBD the calculation of the weighed mean $\overline{\Delta o}$ is the same to that of the DBD.

The calculation of the learning rate, $\alpha_{ij(n)}$, is the following:

$$\textit{if} \quad \left(\overline{\Delta o}_{(n-1)} \cdot \Delta o_{(n)}\right) > 0 \qquad \textit{then} \quad \Delta\alpha_{ij(n)} = k\alpha \cdot \exp\left(\gamma\alpha \cdot \left|\overline{\Delta o}_{(n)}\right|\right)$$

$$\textit{if} \quad \left(\overline{\Delta o}_{(n-1)} \cdot \Delta o_{(n)}\right) < 0 \qquad \textit{then} \quad \Delta\alpha_{ij(n)} = -\psi\alpha_{ij(n)} \cdot \alpha_{ij(n)}$$

$$\textit{otherwise} \qquad\qquad \Delta\alpha_{ij(n)} = 0$$

The calculation of the Momentum's coefficient, $\mu_{(n)}$, is the following:

$$\textit{if} \quad \left(\overline{\Delta o}_{(n-1)} \cdot \Delta o_{(n)}\right) > 0 \qquad \textit{then} \quad \Delta\mu_{ij(n)} = k\mu \cdot \exp\left(\gamma\mu \cdot \left|\overline{\Delta o}_{(n)}\right|\right)$$

$$\textit{if} \quad \left(\overline{\Delta o}_{(n-1)} \cdot \Delta o_{(n)}\right) < 0 \qquad \textit{then} \quad \Delta\mu_{ij(n)} = -\psi\mu_{ij(n)} \cdot \mu_{ij(n)}$$

otherwise $\Delta\mu_{ij(n)} = 0$

It must be noted that the learning coefficient and the Momentum coefficient have different constants that control their increase and decrease.

They are increased according to an exponential and waning function of the absolute value of the gradient's weighed mean; in this way there will be higher increases in correspondence of the areas with small slope and curvature, and minor increases in correspondence to the high curvature areas.

Then, EDBD partially solves the problem of the bounds in the weights' space. Furthermore, in order to ulteriorly reduce the bounds and oscillations in the weights' space for each weight, the following conditions must be verified:

$$\alpha_{(n)} \leq \alpha_{\max}$$

$$\mu_{(n)} \leq \mu_{\max}$$

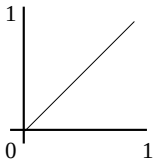
where α_{max} and μ_{max} are the upper limits of the two coefficients.

The Functional Link

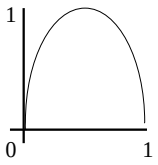
The Functional Link is a process and static expansion technique of the Input values, developed by some Japanese authors (Liu, 1992a, 1992b). During the learning processes, its advantage seems to be that of subjecting the same Input to ANN, from a *series of points of view* that constitute its approximate plausibility.

Giving a value x_i in Input, the Functional Link writes this value again in 6 different ways, expanding so that the original ANN in an ANN provided with an Input vector is 6 times longer. In practice:

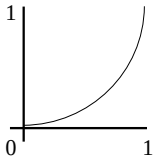
(1) $x_i \rightarrow x_i$ (*identity*)



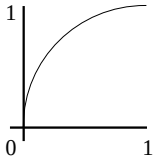
(2) $x_i \rightarrow sin(\Pi \cdot x_i)$ (*sine*)



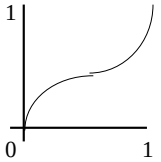
(3) $x_i \rightarrow x_i^{\,2}$ (*Fuzzied AutoIntersection*)



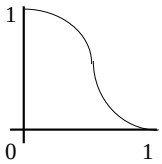
(4) $x_i \rightarrow 2x_i - x_i^{\,2}$ (*Fuzzied AutoUnion*)*



(5) $x_i \rightarrow 4 \cdot (x_i - 0.5)^3 + 0.5$ (*Anti Sigmoid*)



(6) $x_i \rightarrow 0.5 \cdot (\cos(\Pi \cdot x_i)) + 0.5$ (*Inverse Sigmoid*)



Jung's Rule of Semeion

One of the aims of every learning process consists in allowing the layer of the Hidden unit to codify in a viable way for the Input vector.

In order to orient this codification, it has been decided to reinforce those connections between *Input* and *Hidden units* with a larger correction rate, in which the value of these latter are more similar to the Input value from which the connection comes (M. Buscema, 1995, October: experiments at

* Equation (4) has been proposed by Semeion.

Semeion).

This means that the Hidden units modulate their fan-in connections, in relation to their *archetypicity* with the nodes of the Input vector and in relation to the derivative of the error in Output.

We have defined this learning law as *Jung's Rule* (JR), only in virtue of the forcing that the Hidden units suffer in order to codify an "archetype" of the Input vector.

From the algebraic point of view, the JR appears as:

$$(27) \quad Rate'_{ij} = Rate_{ij} \cdot JR_{ij}$$

where: $Rate_{ij}$ = correction rate of the weight w_{ij}; JR = Jung's coefficient.

$$(28) \quad JR_{ij} = \exp\left[\frac{(u_i - u_j)^2}{NumInput} \right]$$

where: u_i = value of i-th Hidden unit; u_j = value of j-th Input unit;
$NumInput$ = number of the Input units.

The JR has offered good results especially for the prediction of *temporal series*. The JR has shown to be able to correctly foresee between 40% and 50%, the sudden tendency changes of a value. Of course, the use of JR implicates a number of Hidden units higher than the usual, because the learning is subjected to a higher number of constraints.

Freud's Rule of Semeion

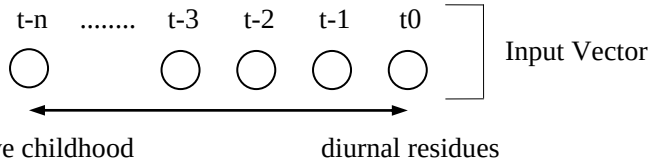
The prediction of temporal series is often associated with the difficulty of anticipating sudden changes of tendency. In BP ANN this problem is evident: the codification of temporal successions through the Input vector (t-n,....t-3, t-2, t-1, t0) involves the Input, which codifies the time previous to the prediction (t0 $\rightarrow$ t+1), and appears as the *most influential* for prediction hypothesis of ANN.

The analysis of sensitivity has shown this phenomenon (F. Matera, 1995, October: experiments carried out at Semeion). This means that often (in about 75% – 85% cases) in the presence of a *critical point*, ANN isn't able to anticipate the tendency change of the temporal series and then it makes predictions in counter-tendency.

The practical seriousness of this phenomenon is obviously linked to the number of critical points that occur in a temporal series. Nevertheless, both theoretically and practically, the critical points of a temporal series are the more important points of a temporal prediction process.

In order to overcome this difficulty, the ANN was provided with a *proto-unconscious* device, so that the experiences which are *more distant in time* affect its learning in a stronger way than those which are more recently closer (M. Buscema, 1995, November: experiments at Semeion).

In brief, this has been done so that the learning of ANN is mostly determined by the Inputs which represent its "relative childhood" and not by those which represent its "diurnal residues":



Freud's Rule (FR) expects that the fan-out connections of each Input node are *much more correct* as the node is farther from the Input node which represents the actual time of the record (t0).

The equation that implements this criterion is very simple:

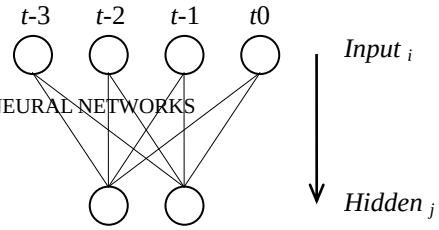
$$Rate' = Rate \cdot F \tag{29}$$

where: *Rate* = correction rate; *F* = Freud's coefficients.

$$F = 1 - \frac{(P_i - 1)}{NumInput} \tag{30}$$

where: P_i = Position of the Input Node (read from the left side), from which the connection comes; *NumInput* = Total Number of the Input Nodes.

Example: giving an Input vector of 4 Nodes and 2 Hidden units, posing *Rate* = 1:



The weights coming from the 4 Input nodes would be correct with the following values:

$$\text{t-3} \rightarrow i = 1 \qquad Rate' = 1 \cdot \left(1 - \frac{1-1}{4}\right) = 1$$

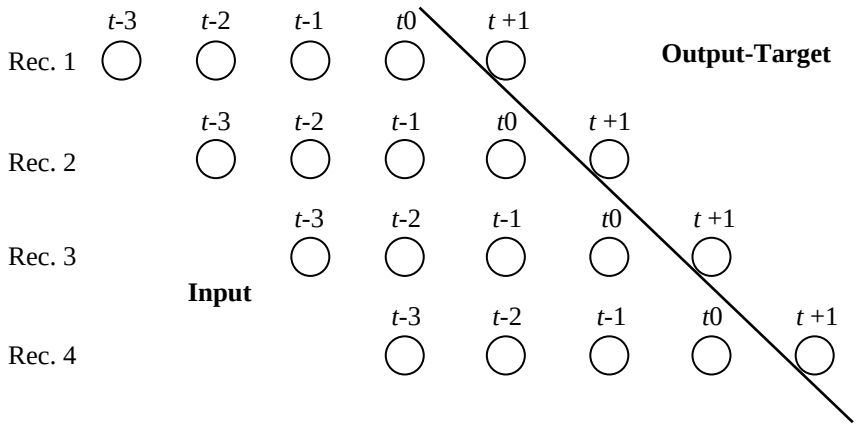
$$\text{t-2} \rightarrow i = 2 \qquad Rate' = 1 \cdot \left(1 - \frac{2-1}{4}\right) = 0.75$$

$$\text{t-1} \rightarrow i = 3 \qquad Rate' = 1 \cdot \left(1 - \frac{3-1}{4}\right) = 0.5$$

$$\text{t0} \rightarrow i = 4 \qquad Rate' = 1 \cdot \left(1 - \frac{4-1}{4}\right) = 0.25$$

Then, with FR we obtain a *linear decrease* of the learning rate of the connections coming from the Input vector in relation to the position of the Input node from which they come.

If FR appears very simple and linear, its consequences on the learning process are otherwise rather complex. In fact, the codification of a temporal series in a BP ANN determines that the succession of the different Records causes each Input node to move from its actual position (t0) to the most remote (t-n):



This means that all Input values of the temporal series end, affecting the learning process in the *same way*, but their influence grows linearly to the decreasing of their actuality.

Through the FR the learning process is much more difficult than the one occurring according to the traditional Back Propagation. Furthermore, the Input vector of ANN changes into a node's level whose position *topographically is meaningful* for the learning itself. This is evidenced at the end of the learning, by carrying out the *analysis of sensitivity* on the resulting weights matrix. This, in fact, shows a specific distribution of the Input nodes on the Output *not being linearly connected* to the position of the Input node in the vector; the most "recent" Input nodes aren't generally those more important for the Output.

Furthermore, on the predictional level, the FR has allowed a meaningful reduction of prediction errors of the tendency changes (critical points). The FR correctly predict between the 45% and the 65% in our experiments on 100 critical points, in respect to the series by utilizing the traditional Back Propagation which is correctly predicted between 15% and the 25% obtained on the same temporal series.

Nevertheless, FR makes some more errors on the predictions that *maintain* the same tendency. Sometimes it invents tendency changes that don't exist. Having provided ANN with a "proto-unconscious" device, it was to be expected that it would "rave" or go "crazy". But globally, the prediction of all types of tendency has demonstrated that the FR is more efficacious than other Back Propagations.

REFERENCES

AA.VV., 1991: AA.VV., *Advanced in Neural Information Processing*, vol. 3, Morgan Kaufman, San Mateo, CA, 1991.

ANDERSON, 1988: J. A. Anderson and E. Rosenfeld, (eds.), *Neurocomputing Foundations of Research*, The MIT Press, Cambridge, Massachusetts, London, England, 1988.

BRIDLE, 1989: J. S. Bridle, *Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition*, in F. Fogelman-Soulié and J. Hérault (eds.), *Neuro-computing: Algorithms, Architectures*, Springer-Verlag, New York.

BUSCEMA, 1993: M. Buscema and G. Massini, *Il Modello MQ*, Collana Semeion, Armando, Rome, 1993. [The MQ Model: Neural Networks and Interpersonal Perception, Semeion Collection by Armando Publisher].

BUSCEMA, 1994: M. Buscema, *Squashing Theory. Modello a Reti Neurali per la Previsione dei Sistemi Complessi*, Collana Semeion, Armando, Rome, 1994 [Squashing Theory: A Neural Network Model for Prediction of Complex Systems, Semeion Collection by Armando Publisher].

BUSCEMA, 1996: M. Buscema, *SQUASH. Shell for program Feed Forward Neural Networks*, Semeion Software n. 5, Rome, 1992–1996.

BUSCEMA, 1997: M. Buscema, F. Matera, T. Nocentini, and P. L. Sacco, *Reti Neurali e Finanza. Esercizi, Idee, Metodi, Applicazioni*, Quaderni di Ricerca, Armando, Rome, n.2 [Neural Networks and Finance. Exercises, Ideas, Methods, Applications, Semeion Research-book by Armando Publisher, n.2].

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

FAHLMAN, 1988: S.E. Fahlman, *An Empirical Study Of Learning Speed In Back-Propagation Networks*, CMV Technical Report, CMV-CS-88-162, 1988.

FREEMAN, 1991: J. A. Freeman and D. M. Skapura, *Neural Networks, Algorithms, Application and Programming Techniques*, Addison Wesley, CNV Series, 1991.

GORMAN, 1988: R. Gorman and T. J. Sejnowski, *Analysis of Hidden Units in Layered Networks Trained to Classify Sonar Targets*, Neural Networks, 1, pp. 76–90, 1988.

JACOBS, 1988: R. A. Jacobs, *Increased Rates of Convergence Through Learning Rate Adaptation*, Neural Network, n.1, pp. 295–307, 1988.

LAPEDES, 1987: A. Lapedes and R. Farber, *Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling*, Los Alamos National Laboratory Report LA-UR-87-2662, 1987.

LIU, 1992a: Q. Liu, S. Hirono, and I. Moriguchi, *Application of Functional-Link Net* in QSAR. 1. QSAR for Activity Data Given by Continuous Variate, in Quant. Struct. -Act. Relat. //, 135–141, School of Pharmaceutical Sciences, Kitasato University, Shirokane, Minato-ku, Tokyo 108, Japan, 1992.

LIU, 1992b: Q. Liu, S. Hirono, and I. Moriguchi, *Application of Functional-Link Net* in QSAR. 2. QUSAR for Activity Data Given by Continuous Variate, in Quant. Struct. -Act. Relat. //, 318–324, School of Pharmaceutical Sciences, Kitasato University, Shirokane, Minato-ku, Tokyo 108, Japan, 1992.

McCLELLAND, 1988: J. L. McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing*, The MIT Press, Cambridge, MA, 1988.

METZGER, 1990: Y. Metzger and D. Lehmann, *Learning Temporal Sequence by Local Synaptic Changes*, in Network 1, pp. 169–188, UK, 1990.

MINAI, 1990: A. A. Minai and R. D. Williams, *Acceleration of BackPropagation through Learning Rate and Momentum Adaptation*, International Joint Conference on Neural Networks, vol. 1, January, pp. 676–679, 1990.

MINSKY, 1954: M. Minsky, *Neural Nets and the Brain-Model Problem*, Doctoral Dissertation, Princeton University, 1954.

MINSKY, 1969: M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA (expanded edition 1988).

MULSANT, 1990: B. H. Mulsant, *A Neural Network as an Approach to Clinical*

Diagnosis, Neural Modeling, vol. 7, n. 1, 1990.
NELSON, 1991: M. McCord Nelson and W. T. Illingworth, *A Pratical Guide to Neural Network*, Addison Wesley, New York, 1991.
NEURALWARE, 1993: NeuralWare, *Neural Computing*, NeuralWare Inc., Pittsburgh, PA, 1993.
NEURALWARE, 1995: NeuralWare, *Neural Computing*, NeuralWare Inc., Pittsburgh, PA, 1995.
ROSENBLATT, 1962: F. Rosenblatt, *Principles of Neurodynamics*, Spartan, N.York, 1962.
RUMELHART, 1986a: D. E. Rumelhart and J. L. McClelland, (eds.) *Parallel Distributed Processing*, Vol.1 *Foundations*, *Explorations in the Microstructure of Cognition*, Vol. 2 *Psychological and Biological Models*. The MIT Press, Cambridge, MA, London, England 1986.
RUMELHART, 1986b: D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*, in RUMELHART 1986a, vol. 1.
RUMELHART, 1986c: D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Back Propagating Errors*, Nature 323: 533–536, in ANDERSON 1988.
SAMAD, 1988: T. Samad, *"Back-Propagation is significantly... "*, International Neural Network Society Conference Abstracts, 1988.
SAMAD, 1989: T. Samad, *Back-Propagation Extension*, Honeywell SSDC Technical Report, 1000 Bane Ave, N., Golden Valley, NN 55427, 1989.
SMITH, 1993: M. Smith, *Neural Networks for Statistical Modeling*, Van Nostrand Reihnold, New York, 1993.
TAWEL, 1989: R. Tawel, *Does Neuron Learn Like the Synapse?*, in TOURETZKY 1989.
TOURETZKY, 1989: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems*, vol. 1, Morgan Kaufman, San Mateo, CA, 1989.
TOURETZKY, 1990a: D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems*, vol. 2, Morgan Kaufman, San Mateo, CA, 1990.
TOURETZKY, 1990b: D. S. Touretzky (ed.), *Connectionist Model*s, Proceedings of the 1990 Summer School, Morgan Kaufman, San Mateo, CA, 1990.
TOURETZKY, 1991: D. S. Touretzky, J. L. Elman, T. J. Sejnowski and G. E. Hinton, *Connectionist Models*, Proceedings of the 1990 Summer School, Morgan Kaufmann, San Mateo, CA.
WEIGEND, 1991: A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, *Back-Propagation, Weight-elimination and Time Series Prediction*, in AA.VV. 1991: pp. 857–882.
WERBOS, 1974: P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences*, Phd Thesis, Harvard, Cambridge, MA, 1974.
WIDROW, 1985: B. Widrow and S. D. Steams, *Adaptive Signal Processing, Signal Processing Series*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

THE AUTHOR

Massimo Buscema, Dr., computer scientist, expert in artificial neural networks and adaptive systems. He is the founder and Director of the Semeion Research Center of Sciences of Communication, Rome, Italy. He is formerly Professor of Science of Communication, University of Charleston, Charleston, West Virginia, USA, and Professor of Computer Science and Linguistics at the State University of Perugia, Perugia, Italy. He is a member of the Editorial Board of *Substance Use & Misuse*, a faculty member of the *Middle Eastern Summer Institute on Drug Use*, co-editor of the Monograph Series *Uncertainty* and co-creator and co-director of *The Mediterranean Institute*. He is consultant of *Scuola Tributaria Vanoni* (Ministry of Finance), *Ufficio Italiano Cambi* (Bank of Italy), *ENEA* (Public Oil Company), *Sopin Group* (Computer Science Corporation) and many Italian Regions. He has published books and articles; among them: *Prevention and Dissuasion*, EGA, Turin, 1986; *Expert Systems and Complex Systems*, Semeion, Rome, 1987; *The Brain within the Brain*, Semeion, Rome, 1989; *The Sonda Project: Prevention from self and heterodestructive behaviors*, Semeion, Rome, 1992; *Gesturing Test: A Model of Qualitative Ergonomics* ATA, Bologna, 1992; *The MQ Model: Neural Networks and Interpersonal Perception*, Armando, Rome, 1993; *Squashing Theory: A Neural Networks Model for Prediction of Complex Systems*, Armando, Rome, 1994; *Self-Reflexive Networks: Theory, Topology, Application*, Quality & Quantity, 29, Kluver Academic Publishers, Dordrecht, Holland; *Idee da Buttare*, Edizioni Sonda, Turin, 1994; *Artificial Neural Networks and Finance*, Armando, Rome, 1997; *A General Presentation of Artificial Neural Networks*, in *Substance Use & Misuse*, 32(1), Marcel Dekker, New York, 1997; *The Sonda Project: Prevention, Prediction and Psychological Disorder*, in *Substance Use & Misuse*, 32(9), Marcel Dekker, New York, 1997.

View publication stats