ON THE PROBLEM OF TRAINING THE COULOMB ENERGY NETWORK

JOHN F. VASSILOPOULOS[†], CRIS KOUTSOUGERAS[‡], and ARTURO HERNANDEZ-AGUIRRE[‡]

Center for Bioenvironmental Research[†]
Tulane University,
New Orleans, LA 70112, USA

Computer Science Department[‡]
Tulane University,
New Orleans, LA 70118, USA

The Coulomb Energy network offers a unique perspective towards nonlinear transformations. However, its training as it was originally proposed by C. Scofield [1] presented difficulties that prevented its general use. We have investigated this model and we present here the reasons for its shortcomings. Further we propose refinements to the model and its training algorithm, and we present the study and results of various other modifications. We address these problems by constraining its architecture (topology) and present a derivation of the associated training algorithm. We also discuss further refinements of this algorithm. Existing genetic algorithms and simulated annealing are also evaluated as training techniques. Simulation results are also presented.

1. Introduction

The problems which neural networks address can generally be formulated as nonlinear function approximations, associative memories, clustering mechanisms, or probability density mappings. The outcome of such methods depends on responses to new inputs and the way they are interpolated from already known samples. An interesting problem which we take up in this study does not fall easily in any of these categories. We examine the problem here and discuss its formulation in the neural networks context. We shall provide the mechanism for discovering a mapping with certain aggregate quality measures as opposed to a mapping that simply attains data fitting.

The quality of the mapping shall not be assessed by the fitting of target values per se, but rather by its overall topological properties. Although a sample of output is provided in a manner comparable to supervised methods, there is no predefined target output to which the neural network should establish a curve fitting or a clustering based on minimization of error. The approach presented in the following sections establishes a blending of supervised and unsupervised methods [2] where the unsupervised component aids and controls the development of internal representations by employing similarity properties among the input data.

The reason we are exploring a mapping with controlled aggregate quality measures, or controlled topology, is because this implies control of the neural network's internal representations. The development of internal representations in the more traditional curve fitting methods is more of a side effect or a fortunate consequence. By exercising control on the development of the internal representations we expect more robust training that builds more accurate "world models" that better capture the significant properties and dynamics of the process to be approximated.

The philosophical foundation behind our approach is the recognition of the fact that when one attempts to learn the secrets which unfold the dynamics of an observed process, the first steps are to identify the unique properties of the process and its structure. The actual target values are not as important in the first steps, but rather their usefulness lies in what they describe about the structure of the process, such as symmetries and input space areas that are correlated or where their behavior is identical. Then, only the fine-tuning step of attaining exact match of target values follows. As a simplistic example, consider the case where the function F to be learned behaves identically in two disjoint areas A_1 and A_2 of the input space. One scenario that may develop (common with existing networks) is that the network learns the behavior of F independently in A_1 and in A_2 . Although the target outputs may be fitted perfectly, in this case the network may not reflect in its internal representational structure this topological property associated with the function F. Another scenario (which is what we desire) is that the network may develop such an internal representational structure that will remap the original input space into an intermediate one where A_1 and A_2 are both remapped and folded into the same region of the intermediate space A_{12} . Then the behavior of F is learned only in A_{12} instead of using both A_1 and A_2 to attain the same task. In such a case the network would reflect in its internal representational structure the fact that A_1 and A_2 are identical as far as F is concerned. As it shall be analyzed, if the only target of the adaptation is the fitting of the target values, the first scenario is more likely and the second one simply a fortunate coincidence. The above concept is an example of the type of space manipulation that we desire, however it is not the direct target of this work; we simply would like to capitalize on this concept for the purposes of learning mappings with controllable aggregate properties and more particularly for the purposes of pattern recognition problems.

The way the pattern recognition problem has been traditionally approached with neural nets is as follows. In one approach the net identifies and reflects the borders of each class in the input space; this is the clustering approach. In another approach, arbitrary identifiers are assigned to classes, and these identifiers become the target values for curve fitting; thus, a piecewise constant function is approximated by a continuous function that is implemented by a neural net. As we mentioned, even in the second approach the net is more likely to favor a clustering of the input space in its internal representation and then approximate the target values in each cluster. Our goal is to achieve a transformation of the input space into a new space via a continuous mapping so that (i) any two inputs of same class map on outputs that are very close, and (ii) any two inputs of different class map on outputs that are significantly apart. Closeness is measured by any distance metric but the Euclidean distance is chosen here. There are no specific target values that are set apriori. A class can then be represented (identified) by the mean output value for inputs from this class and such identifiers would be quite distinct for various classes if the above properties (i) and (ii) hold. The idea is to transform the input space so that the similarity or dissimilarity of various vectors is reflected in their distance and thus in the topology of the transformed space. This would naturally lead to a mapping that would unify/identify two input space regions of same class by mapping them in the same region of the output space.

A net that would achieve this would have to develop internal representations that would reflect the fact that these two regions become indistinguishable as far as the classification process at hand is concerned. What is important in this approach is that the net is forced to identify or discover such structural properties of the classification process, whereas in existing systems this 'discovery' is more of a coincidence that may or may not occur.

In the following section we describe the fundamentals of this type of formulation and we analyze another relevant attempt. One of the contributions of this treatise is to provide the insight behind the deficiencies of this existing attempt by further reformulating the problem and deriving a new solution.

Our goal is to transform, in the Hadamard sense [3], an ill-posed problem where there are several non-unique solutions that do not depend continuously on the data, to a well-posed problem where there exists a unique solution that depends continuously on the data and more significantly does not yield unbounded output errors based on initial conditions. We have studied and developed an understanding of the problem and its intricacies and we present an insight to its behavior through experimentation that ultimately incorporates a formulated solution. We also produce the basic solution derivation and we evaluate the performance of this basic method and submit experiment results.

2. Background

A work that similarly addresses the problem is the supervised learning algorithm for the N-dimensional Coulomb network which is applicable to multi-layer networks and has been developed by Christopher L. Scofield [1]. The central idea was to define a potential energy of a collection of memory sites where each site played the role of an attractor for other similar sites or a repeller for dissimilar sites. The target was to minimize the energy for the entire memory site collection at each layer of a multi-layer network. It is essential to note that this optimization process was performed in succession at each layer independently of other layers. Unlike backpropagation [4], there is no dependency on the propagation of error, thus each layer can be trained independently. The definition of attractive potential energy between memory sites of the same class and repulsive potential energy between memory sites of different class is given by the following two conditions:

$$\begin{aligned} \mathrm{sign}(Qi(c)) &\neq \mathrm{sign}(Qi(c')) \quad \text{for } c = c', \\ \mathrm{sign}(Qi(c)) &= \mathrm{sign}(Qi(c')) \quad \text{for } c \neq c', \end{aligned} \tag{1}$$

where *Qi(c)* is the charge at memory site *i* of class *c*. This ensured that memory sites of same class were assigned different sign, therefore attracting each other, and memory sites of different class were assigned the same sign, therefore repelling each other.

In absence of the test charge, the electrostatic potential energy for the configuration of memory sites should be:

$$\psi = \frac{1}{2L} \sum_{i=1}^{M} \sum_{j=1}^{M} Q_i Q_j \left| x_i - x_j \right|^{-L} \tag{2}$$

The following logistic function, discussed extensively by Rumelhart and others [4], was employed:

$$F_n\left(\sum_{m=1}^{K} w_{nm} f_{mi} \right) = \left(1 + \exp(-(1/kT)\left(\sum_{m=1}^{K} w_{nm} f_{mi} + \Theta \right)) \right)^{-1} \tag{3}$$

where the n-th cell has activity F_n , w_{nm} is the matrix of afferent synapses, f_i is the afferent activity in space R^K , and x_i is the vector of network activity from mapping f_i via w_{nm}. The memory sites move in space as a result of the different charges and the target is to minimize the electrostatic energy ψ by computing the gradient of the potential energy with respect to the weights w_{nm}:

$$\delta w_{nm} = -\frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} Q_i Q_j \left| x_i - x_j \right|^{-(L+2)} \Delta_{nm}(f_i \ , f_j) \tag{4}$$

Each node in the network must 'remember' the locations of memory sites in its activity space in order to be able to compute their differences ($R_{ij} = x_i - x_j$), or the outcome could be approximated by computing each term and accumulating its value along with other terms in a temporal fashion.

The problem with the above formulation is that it assumes training of one layer at a time. Layers are added as needed to eventually conclude with a multi-layered network that would accomplish the mapping with the desired topological properties. However, it is not obvious whether this process has any guarantees that it can always accomplish this task. In multi-layered networks there is usually a synergism among the various layers. The manner in which a certain layer is adjusted is in direct or indirect relation to what happens in other layers. In more intuitive terms, the way the functionality of a layer is adjusted depends on how well it can make use of the signals/features that are passed to it from lower layers and how useful this layer's functionality proves to be for other layers above it. This is actually what makes Backpropagation such a powerful adaptation scheme; adjustments of local functionality ensure better global performance by taking into account how this local functionality uses and is used by the rest of the features that are synthesized by lower layers, and it combines them to synthesize higher order features which then can be used by upper layers in synthesizing the appropriate concepts. Thus, given a network of fixed topology and an overall task, more traditional training algorithms (such as Backpropagation) assign subtasks to the various network resources which in turn collaborate at different levels (neurons, layers) to achieve the overall task. If a network topology is not fixed, this allocation is a rather challenging task.

The fact that a layer can synthesize or add a new feature may not be useful if there is no indication of how it can further be used (this indication is passed down - backpropagated - from higher layers). The only evidence to this effect is whether the new feature is directly useful in achieving the goal task at the current layer. If this feature could be indirectly useful to the goal task at a latter stage (after the addition of more layers), then its significance may not be evident at the current layer and it may never be synthesized.

In conclusion, many necessary features cannot be synthesized by using a single layer but can be synthesized by using two layers [5]. Moreover, it is not obvious that a necessary feature that needs two layers will actually be synthesized in a piecemeal fashion by adjusting one layer first and then adding and adjusting a second layer as it is envisioned in Scofield's approach. Likewise, it is not apparent why it is always possible to synthesize this feature that is ultimately useful in optimizing a function G by optimizing G at one layer (and partially constructing the feature) and then completing the feature in the next layer by again optimizing the same function G. This would be possible only if the overall function G to be optimized can be synthesized by a cascade of other functions $G = G_1 \circ G_2 \circ G_3 \circ ... \circ G_n$ that happen to have the property that each G_i can be implemented by a single layer, and G can be optimized by optimizing G_i's in a fixed sequence of 1...n. In the case of the network used by Scofield, G is not fixed. At each layer we have a partial $G_1 \circ G_2 \circ G_3 \circ ... \circ G_k$ and we look for G_{k+1} , G_{k+2} , ... until the current overall G conforms to the requirements (becomes optimized).

The strong constraint though is that each of the G_i's must be realizable with a single layer. Thus, this approach may become troubled when we are constrained to use a G_k that is realizable by a single layer so that $G_1 \circ G_2 \circ G_3 \circ ... \circ G_k$ is better optimized than $G_1 \circ G_2 \circ G_3 \circ ... \circ G_{k-1}$. It is therefore recognized that we need to optimize at least two layers at a time instead of one. Given that this is the case, it turns out that we only need two layers and there is no real necessity to employ more than two. Funahashi [5] has proven that two layers are sufficient to approximate any function.

Thus, a function that implies a desired transformation is realizable with two layers and we only need to consider a two layered network. Our goal then becomes to reformulate the problem in the two layer perspective and derive the learning algorithm within this perspective since from the above discussion it follows that Scofield's solution cannot be directly extended.

3. Problem Formulation

We assume that we are given an n-dimensional space S_n in which an arbitrary but finite set of vectors are labeled with labels from another finite set C. This represents an object space with some objects classified in a set C of classes. We wish to identify another m-dimensional space S_m into which S_n is mapped through some nonlinear scheme. A condition imposed on this mapping is that objects from S_n with same labels are mapped on the same object of S_m and that any two objects of S_n with different labels are mapped on different objects of S_m (Figure 1). This should be accomplished by a nonlinear mapping function which is analytic (continuous and differentiable) in order to be able to provide a tool for interpolation and to be realizable using neural networks.

The goal is to develop a mechanism for the automatic discovery of this mapping function. Alternatively, the problem can be reformulated by introducing a measure of similarity among labels and requiring that the Euclidean distances of the images X_1' and X_2' (in S_m) of two vectors X_1 and X_2 (in S_n) reflect the similarity of the labels of X_1 and X_2. Thus, we would like the inverse Euclidean distance in S_m to reflect the similarity of the labels of the parent vectors in S_n.

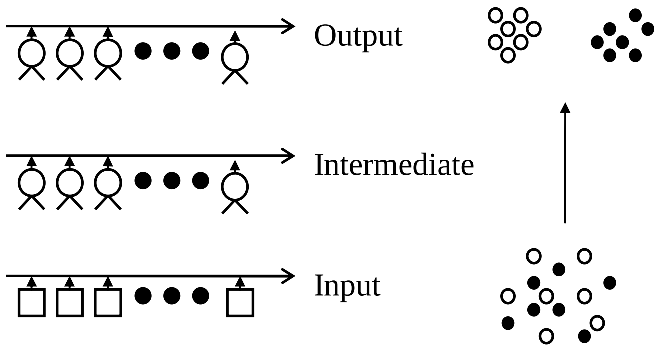


Figure 1 Successive intermediate transformations

This is a formulation of a pattern classification problem since the nonlinear function can be used to classify the vectors of S_n. Traditionally, this problem is approached with clustering schemes which partition S_n in clusters of similar vectors. The Self-Organizing-Map (SOM) network [6] is one that uses the idea of competitive filtering or learning where only winning neuron nodes and their neighbors are scheduled to change their weights. It has also been approached with function approximators where labels are numerically quantified and used as (one-dimensional) targets in supervised learning schemes like backpropagation. In such cases, the space S_m is essentially fixed a priori and then one tries to identify the mapping function which achieves it. In the above formulation such targets are not assumed; the space S_m is to be identified, and then tied to the identification of the mapping function.

Thus, the difference here is that we are looking for a mapping function that imposes a certain aggregate property on the topology of the space S_m , which it implies, but S_m is not predefined. Backpropagation's main constraint is that it is considering fixed targets against which it must minimize the error.

In the more traditional approach of non-linear regression networks, the guide for the adaptation is the fitting of the target values. The network is tuned to reproduce the behavior of an observed process at a finite set of input instances (training set). Naturally, the network implements a function that is defined on the entire input space. All that the adaptation ensures is that the network's function and the observed process coincide at the sample set. It is only hoped that the learned function approximates closely the observed process on the entire input space and that the network has thus become a simulator of the observed process. The network (internally and incrementally) synthesizes the function it represents. At the output layer of the net we observe the overall result of the adaptation and the network's overall function. Internally, the network has developed complex representations and descriptions about properties that make this overall function describable. Therefore, if the network's function is a proper simulation of the observed function, then the network's internal representations must somehow capture or reflect the properties of the observed process or the dynamics of the physical system that generates this observed process. Based on the fitting criterion alone, the development of proper internal representations is a secondary matter that is handled only indirectly by the adaptation process.

With our formulation, direct attention is paid to the internal representations because we are not only concerned with the discovery of a mapping that fulfills local constraints (fitting the samples) but we also target aggregate properties of the mapping as a whole. Hence, there is a better expectation that the neural network will develop proper internal "world models" (Figure 2) for the process it is supposed to simulate.

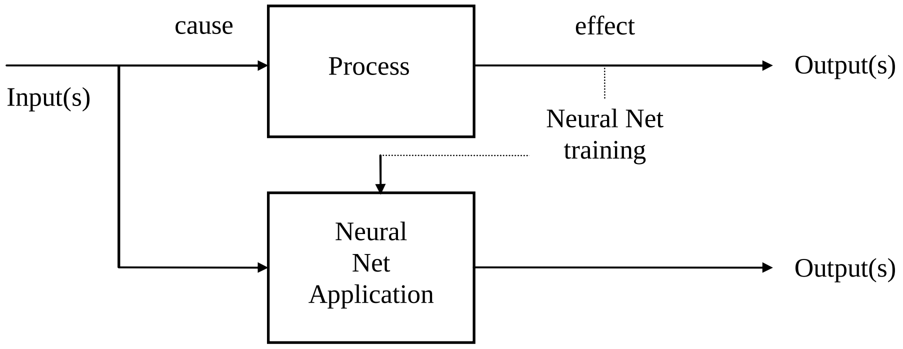


Figure 2 Neural Network as a "World Model "

Another distinct difference is that our approach considers pair-wise interactions between vectors where neither vector in the pair is designated as a target. In addition, and by diverging from neural network schemes that use either clustering mechanisms or associative memories [7] as in the Hopfield network [8], neither competitive learning [9] nor association or correlation of stored information will be used. However, we are nonetheless looking for an analytic function and neural networks can elegantly represent parametric (thus adjustable) nonlinear functions, thus we shall employ neural networks as the underlying structures.

4. Problem Analysis

It has been proven that a feed forward net with at least two layers and nonlinear neuron activation functions can approximate any mapping [5]. Thus, we use such a network and the target then becomes the learning algorithm for the weight adjustment. In order to embed the target properties of the mapping in the learning algorithms, we need to express them in a closed form expression representing the "quality" of the mapping. A number of such expressions involving the differences of various net outputs was attempted. One such example was:

$$\max_{w} \sum_{ij} a_{ij} \left\| y_i - y_j \right\|^2 \quad \text{where} \quad a_{ij} = \begin{cases} 1 & label(y_i) \neq label(y_j) \\ -1 & label(y_i) = label(y_j) \end{cases} \tag{5}$$

The rationale supporting the above quality measure is that the Y vectors are parametric in w (the weights); as weights change across the net so do the vectors. The goal then becomes to derive the algorithm that optimizes the quality function. We utilized the generalized derivation for two layer nets but ascertained that functions of the above form do not work for the following reasons: (i) they become unbounded and the simulators can easily crash with floating point exceptions, and (ii) there exist problems due to favoring of dominating terms (points that are close to each other tend to move very little, whereas points that are moving away from each other are doing so under forces that are considerably stronger). Another form was devised to alleviate the aforementioned problems:

$$\max_{w} \sum_{ij} sigmoid \left(a_{ij} \left\| y_i - y_j \right\|^2 \right)$$

$$\text{where } a_{ij} = \begin{cases} 1 & label(y_i \neq label(y_j) \\ -1 & label(y_i = label(y_j) \end{cases} \tag{6}$$

However, this solution encounters the problem of sensitivity in saturation regions (Figure 3) where the derivative is almost zero thus not contributing towards the movement of vectors.

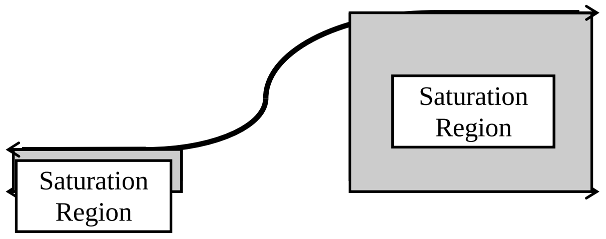


Figure 3 Sigmoid Function Saturation Regions

One form that does not encounter any of the problems that plague the previous approaches is:

$$\min \sum_{ij} \frac{a_{ij}}{\left| y_i - y_j \right|^2}, \qquad \text{where } a_{ij} = \begin{cases} 1 & label(y_i) \neq label(y_j) \\ -1 & label(y_i) = label(y_j) \end{cases} \tag{7}$$

The evidence we have gathered thus far suggests that this is the only form that can be employed, although it is possible to combine it with other forms in an interleaved fashion. The issue this expression presents is that it has local minima which manifest themselves as follows: while weights change the input-output mapping changes and the image vectors in S_m move around in a continuous rearrangement. Thus, the motion reaches a stage where it has to overcome an energy barrier (Figure 4).

If $\begin{cases} y_i = \left(y_{1i}, y_{2i}, \ldots, y_{ni}\right) \\ y_j = \left(y_{1j}, y_{2j}, \ldots, y_{nj}\right) \end{cases}$ then $\dfrac{a_{ij}}{\left|y_i - y_j\right|^2} = \dfrac{a_{ij}}{\sum_k \left(y_{ki} - y_{kj}\right)^2}$

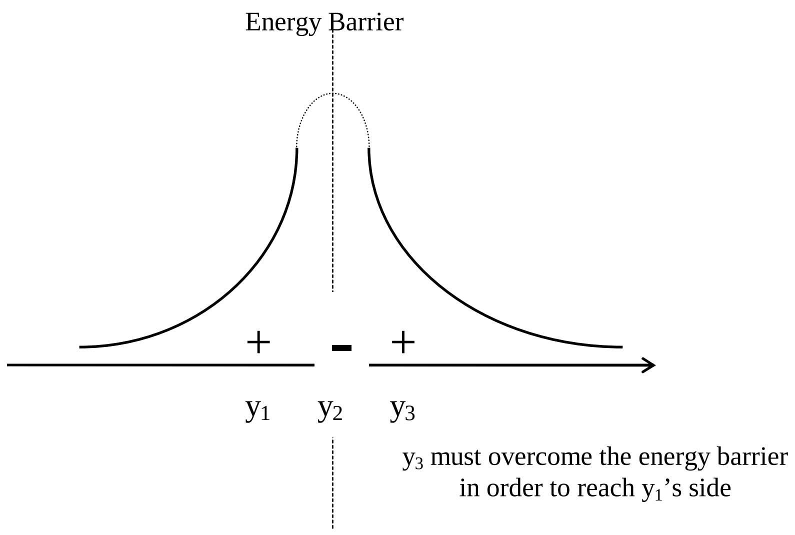

Energy Barrier
+
-
+
y1
y2
y3
y3 must overcome the energy barrier in order to reach y1's side

Figure 4 Energy Barrier Encountered During Movement

This happens due to the differences between corresponding components (being the derivative of terms involving Euclidean distances) combined with the monotonicity of the sigmoid activation function. A number of approaches have been explored and a working model has been constructed to address the above issues as illustrated in the following.

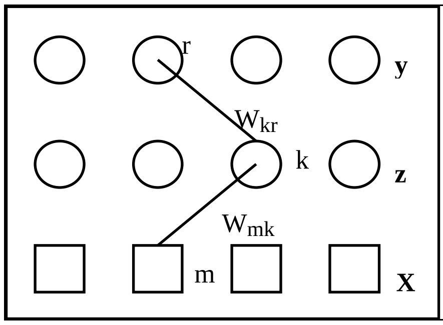
5. Basic Solution Derivation

For reference we will use the legent of figure 5. According to this:

y	represents the output of a neuron at the output layer.
r	is the index of the r-th neuron at the output layer
k	is the index of the k-th neuron at the hidden layer
z	is the output of a neuron at the hidden layer
Z	is the output vector of the hidden layer
m	is the index of the m-th component of the input vector
i	is the the index of a sample vector from the training set
s	is the total net stimulus to a neuron

Thus y_{ir} is the output that the r-th neuron is producing when the i-th sample vector is input to the net. Similarly s_{ik} is the total net stimulus to the k-th neuron when the i-th sample vector is input to the net.

We need to optimize:
$$G = \sum_{ij} \frac{a_{ij}}{| y_i - y_j |^2} \tag{8}$$



Notes:
X = input vector, y = output layer,
z = intermediate/hidden layer,
X_i is the i-th input vector in the training set,
X_{ij} is the j-th component of X_i,
W_{ij} is the weight from neuron i to neuron j.

Figure 5 Neural Network Layer Interfacing

To compute the change in weights we use the gradient of G: $\Delta w = -\alpha \nabla_w G$

From (8) follows:
$$\nabla_w G = \frac{\partial G}{\partial w} = \sum_{ij} a_{ij} \frac{\partial \left(\frac{1}{| y_i - y_j |^2} \right)}{\partial w} \tag{9}$$

We observe that:

$$\frac{\partial \left(\frac{1}{| y_i - y_j |^2} \right)}{\partial w} = \frac{-\frac{\partial \left(| y_i - y_j |^2 \right)}{\partial w}}{| y_i - y_j |^4} = -2 \frac{\frac{\partial \left(| y_i - y_j | \right)}{\partial w}}{| y_i - y_j |^3} \tag{10}$$

Now since:
$$\left| y_i - y_j \right| \equiv \sqrt{\sum_r (y_{ir} - y_{jr})^2} \tag{11}$$

we obtain:
$$\frac{\partial(| y_i - y_j |)}{\partial w} = \frac{\partial \sqrt{\sum_r (y_{ir} - y_{jr})^2}}{\partial w} = \frac{1}{2} \frac{\frac{\partial \sum_r (y_{ir} - y_{jr})^2}{\partial w}}{\sqrt{\sum_r (y_{ir} - y_{jr})^2}} \tag{12}$$

We also have:
$$\frac{\partial \sum_r \left(y_{ir} - y_{jr}\right)^2}{\partial w} = 2\sum_r \left(y_{ir} - y_{jr}\right) \frac{\partial \left(y_{ir} - y_{jr}\right)}{\partial w} \tag{13}$$

From (9), (10), (12), and (13) we get:

$$\frac{\partial G}{\partial w} = \sum_{ij} [a_{ij} \left(\frac{-2}{\left| y_i - y_j \right|^3} \right) \left(\frac{\sum_r \left[(y_{ir} - y_{jr}) \frac{\partial (y_{ir} - y_{jr})}{\partial w} \right]}{\sqrt{\sum_r (y_{ir} - y_{jr})^2}} \right)] =$$

$$-2\sum_{ij} [a_{ij} \frac{\sum_r \left[(y_{ir} - y_{jr}) \frac{\partial (y_{ir} - y_{jr})}{\partial w} \right]}{| y_i - y_j |^3 \sqrt{\sum_r (y_{ir} - y_{jr})^2}}] \tag{14}$$

In the following, we recompute (14) for each network layer

Derivation of update rule for w = w$_{kr}$ (intermediate to output layer)

If $s_{ir} = W_r{}^t Z_i$ is the total input to the r-th output neuron when X$_i$ is the input vector, and if *f* signifies the neuron squashing function, then :

$$\frac{\partial y_{ir}}{\partial w_{kr}} = \frac{\partial f(s_{ir})}{\partial w_{kr}} = \frac{\partial f(s_{ir})}{\partial s_{ir}} \frac{\partial s_{ir}}{\partial w_{kr}} = \frac{\partial f(s_{ir})}{\partial s_{ir}} z_{ik} \tag{15}$$

since
$$\frac{\partial s_{ir}}{\partial w_{kr}} = \frac{\partial \left(W_r{}^t Z_i \right)}{\partial w_{kr}} = z_{ik} \tag{16}$$

If the sigmoid is used as the neuron squashing function: $f(s) = \frac{1}{1+e^{-s}}$ then

$\frac{\partial f(s)}{\partial s} = f(1-f)$ and so by (15) we have:
$$\frac{\partial y_{ir}}{\partial w_{kr}} = y_{ir}(1-y_{ir})z_{ik} \tag{17}$$

Similarly:
$$\frac{\partial y_{jr}}{\partial w_{kr}} = y_{jr}(1-y_{jr})z_{jk} \tag{18}$$

Thus, for weights of connections from the intermediate layer to the output layer, it follows from (14), (17), and (18) that the update rule is:

$$\boxed{\frac{\partial G}{w_{kr}} = -2\sum_{ij} a_{ij} \frac{\sum_r \left[(y_{ir} - y_{jr})(y_{ir}(1-y_{ir})z_{ik} - y_{jr}(1-y_{jr})z_{jk}) \right]}{|\, y_i - y_j \,|^3 \sqrt{\sum_r (y_{ir} - y_{jr})^2}}} \tag{19}$$

Derivation of update rule for w = w

Derivation of update rule for $w = w_{mk}$ (input to intermediate layer)

$$\frac{\partial y_{ir}}{\partial w_{mk}} = \frac{\partial y_{ir}}{\partial z_{ik}} \frac{\partial z_{ik}}{\partial w_{mk}} \tag{20}$$

Now
$$\frac{\partial y_{ir}}{\partial z_{ik}} = \frac{\partial f(s_{ir})}{\partial s_{ir}} \frac{\partial s_{ir}}{\partial z_{ik}} = \frac{\partial f(s_{ir})}{\partial s_{ir}} w_{kr} \tag{21}$$

and
$$\frac{\partial z_{ik}}{\partial w_{mk}} = \frac{\partial f\left(s_{ik}\right)}{\partial s_{ik}} \frac{\partial s_{ik}}{\partial w_{mk}} = \frac{\partial f\left(s_{ik}\right)}{\partial s_{ik}} x_{im} \tag{22}$$

Therefore, from (20), (21) and (22) it follows:

i-th component:
$$\frac{\partial y_{ir}}{\partial w_{mk}} = \frac{\partial f(s_{ir})}{\partial s_{ir}} \frac{\partial f\left(s_{ik}\right)}{\partial s_{ik}} w_{kr} x_{im} \tag{23}$$

Now, if the neuron's squashing function is the sigmoid:

$$\frac{\partial y_{ir}}{\partial w_{mk}} = (y_{ir}(1 - y_{ir})z_{ik}(1 - z_{ik}))w_{kr} x_{im} \tag{24}$$

Similarly for the j-th component:

$$\frac{\partial y_{jr}}{\partial w_{mk}} = (y_{jr}(1 - y_{jr})z_{jk}(1 - z_{jk}))w_{kr} x_{jm} \tag{25}$$

Thus, for weights of connections from the input layer to the intermediate layer, it follows from (14), (24), and (25) that the update rule is:

$$\frac{\partial G}{w_{mk}} = -2\sum_{ij} a_{ij} \frac{\sum_r \left[(y_{ir} - y_{jr}) \left[(y_{ir}(1-y_{ir})z_{ik}(1-z_{ik}))w_{kr}x_{im} - (y_{jr}(1-y_{jr})z_{jk}(1-z_{jk}))w_{kr}x_{jm} \right] \right]}{|\, y_i - y_j \,|^3 \sqrt{\sum_r (y_{ir} - y_{jr})^2}} \tag{26}$$

Conclusively, equations (19) and (26) are used in $\Delta w = -\alpha \nabla_w G$ to obtain the weights update rule for intermediate-to-output and input-to-intermediate layers respectively.

6. Evaluation of the Basic Solution

An algorithmic representation of the neural network simulator was constructed to assist the process of experimentation and aid the theoretical analysis. We have applied this network learning paradigm to the XOR problem and other arbitrary and combined topologies to study its behavior and employ further refinements. The principal reasons for employing the XOR problem are:

- it has been used as a standard benchmark problem in classification networks,
- it poses difficulty due to its symmetry that leads to dynamic equilibrium that plagues gradient descent algorithms.

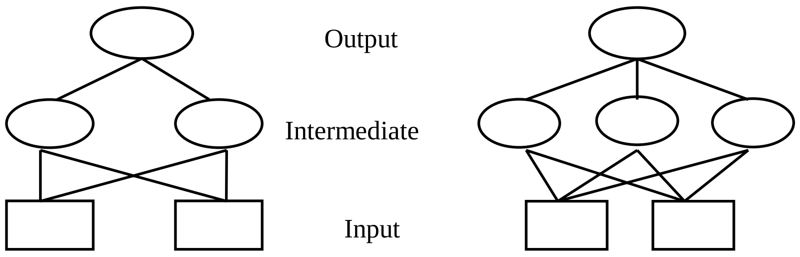


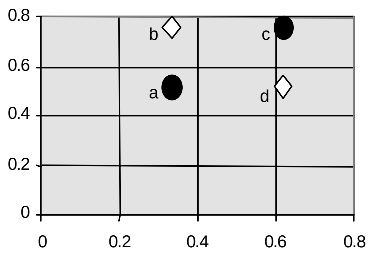
Figure 6. Neural Network Structures with 2 and 3 intermediate neurons.

Several configurations of neurons were used for the intermediate layer, and the two most efficient (Figure 6) in terms of iterations, convergence, complexity, and execution time were utilized. The neural network structure with two intermediate neurons handled all cases of XOR and similar but shifted vector positions, whereas the three intermediate neuron network structure handled all the arbitrarily distributed vector positions.

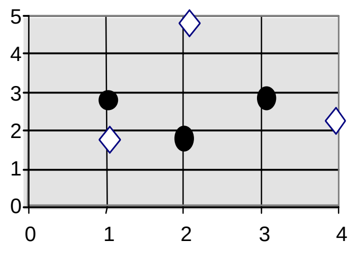
The preliminary run data sets (Table 1) indicated that the model is sensitive to initial conditions. The simulator behavior had to be handled by introducing optimizations methods. The following two sets of graphs (7a, 7b and 8a, 8b) clearly illustrate the network behavior and its ineptitude to always remap the vectors according to our expectations.

	XOR (4 vectors)	**Arbitrary (4 vectors)**	**Arbitrary (6 vectors)**
Number of Iterations Per Run	1000, 10000, 20000, 100000, 200000	1000, 10000, 20000, 100000, 200000	1000, 10000, 20000, 100000, 200000
Generated Runs Distinct sets of initial conditions (weights and threshold values)	100	100	100

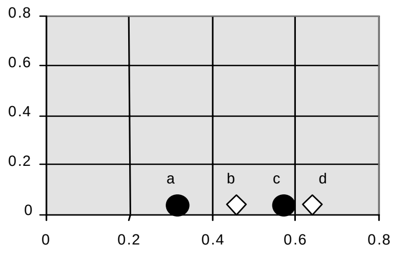
Table 1 Basic Method Runtime Cases



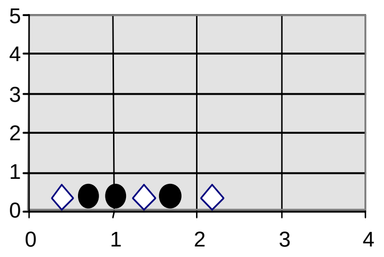
7a Initial Vector Positions (XOR)



8a Initial Vector Positions (arbitrary case)



7b Final Vector Mapping (XOR)



8b Final Vector Mapping (arbitrary case)

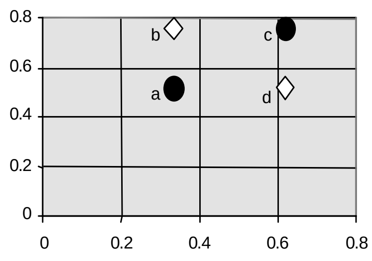
Sample data sets taken from simulator runs employing the basic method for sets of 4 vectors (XOR) and 6 vectors (arbitrary), with 2 distinct classes of objects in each set. We should note that these are cases that present an almost perfect symmetry that leads to dynamic equilibrium that manifests itself as local minima in which the gradient decent gets stuck. So this basic algorithm can be used but it would need the known tricks used to get the gradient decent out of local minima (momentum terms, simulated annealing/temperature parameter, etc.). Because of this sensitivity we attempted a number of refinements as explained in the next section.

7. Further Refinements

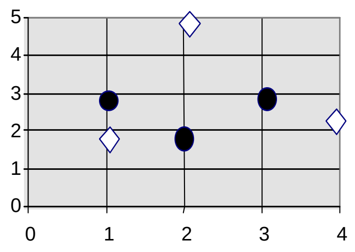
To assess the pattern clustering and the speed of the net, and to refine the basic algorithm further, we adopted the following design modifications and additions:

- adjusted the network in such a way that either attracting or repelling forces were at first turned off, and then gradually increased, thus moving entire clusters of vectors instead of one vector at a time.

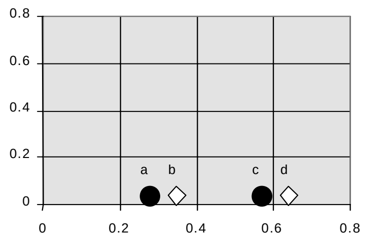
Although separation among groups of vectors was achieved in a prompt manner, the distance among the groups was inhibitive towards remapping individual vectors to the cluster(s) in which they should have membership. The following two sets of graphs (9a, 9b and 10a, 10b) clearly illustrate the network behavior and its inability to remap the vectors in such a way that proper clustering is achieved.



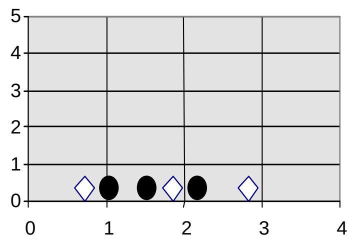
9a Initial Vector Positions (XOR)



10a Initial Vector Positions (arbitrary)



9b Final Vector Remapping (XOR)

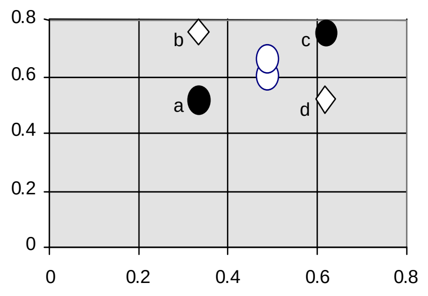


10b Final Vector Mapping (arbitrary)

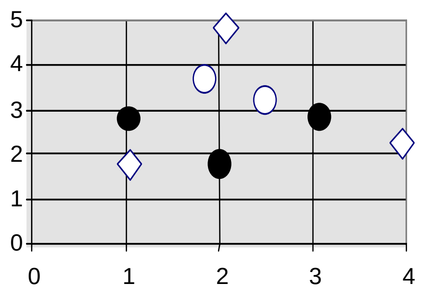
Sample data sets taken from simulator runs employing the attraction pole method for sets of 4 vectors (XOR) and 6 vectors (arbitrary), with 2 distinct classes of objects in each set.

- we used arbitrary poles of attraction for various subsets of the electrostatic charges corresponding to intermediate vectors thus momentarily reverting to supervised methods to break ties created by dynamic equilibrium.

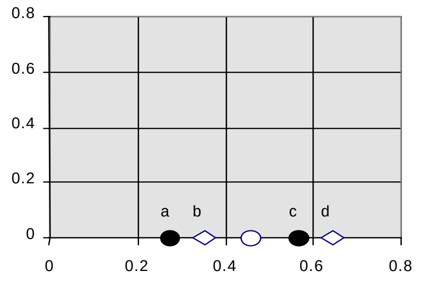
This could be very efficient since interactions between groups of vectors would be considered instead of vector to vector interactions. The poles of attraction were defined statistically, before the net was trained, and their relative positions were static at the early stages but moved during latter steps. We calculated the median vectors and assigned them the role of attraction and/or repulsion. The behavior of the network changed rather negligibly (11a, 11b and 12a, 12b), still unable to overcome ties created by dynamic equilibrium.



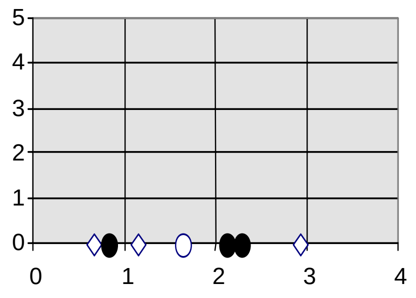
11a Initial Vector Positions (XOR)



12a Initial Vector Positions (arbitrary)



11b Final Vector Remapping (XOR)

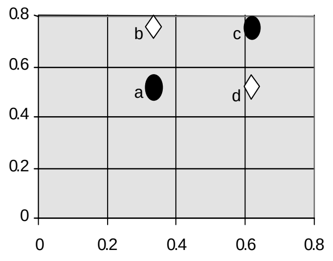


12b Final Vector Mapping (arbitrary)

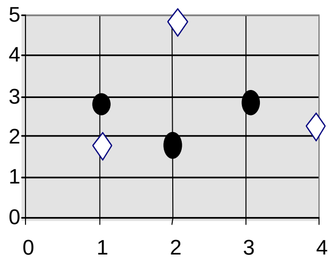
Sample data sets taken from simulator runs employing the attraction pole method for sets of 4 vectors (XOR) and 6 vectors (arbitrary), with 2 distinct classes of objects in each set.

- we introduced a temperature parameter in order to control the steepness of the sigmoid function.

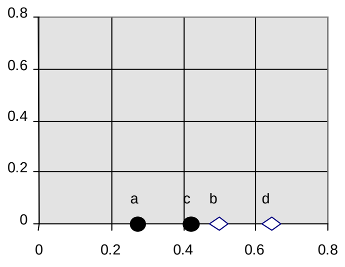
The results of our experimentation were particularly consistent. The manner in which vectors moved towards their respective and final remapped positions was not only efficient, but also expeditious. The temperature variations (in order of magnitudes) determined the number of network iterations for reaching a final stage of grouping. For certain weight measures where minimal (< 1,000) iterations were needed, the temperature value remained at 1, thus not affecting the curvature of the sigmoid. When a set of weights was assigned that required iterations in the order of several thousands (> 1,000 but < 100,000) to achieve the desired grouping, then the temperature value assigned was within the range of 0.1 and 10.0 (excluding 1), thus changing the shape of the sigmoid curve to a steeper or smoother path. Most of the simulator runs did not require adjustment of the temperature variable as they quickly achieved equilibrium with successful vector groupings. The minority of runs that required temperature adjustments were subsequently introduced to temperatures that started with static values and progressively were dynamically decreased or increased during execution. This scheme achieved grouping by minimizing the number of pair-wise interactions between candidate vectors. The following two sets of graphs (13a, 13b and 14a and 14b) clearly display the final vector positions as they cross over to form the expected groups.



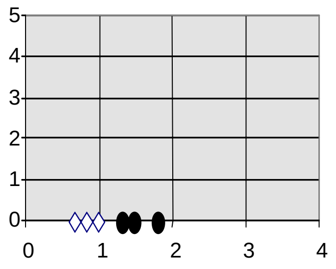
13a Initial Vector Positions (XOR)



14a Initial Vector Positions (arbitrary)



13b Final Vector Mapping (XOR)



14b Final Vector Mapping (arbitrary)

Sample data sets taken from simulator runs employing the temperature method for sets of 4 vectors (XOR) and 6 vectors (arbitrary), with 2 distinct classes of objects in each set.

- finally, we used an implementation of a Simple Genetic Algorithm [10] to assist in accelerating the clustering of vectors by searching for group(s) of initial weight and threshold values that could potentially start the neural net at a more advantageous position than one that would be by arbitrary selection. The neural net was altered to accept as input a static preselected set of input vectors and an initial set of weights and threshold values and produce only one quality value that was introduced as the target value to be minimized by the SGA. Subsequently, the genetic algorithm produced weight and threshold values that were continuously fed back to the neural net which in turn would produce the quality value sent to the genetic algorithm for minimization (Figure 15).

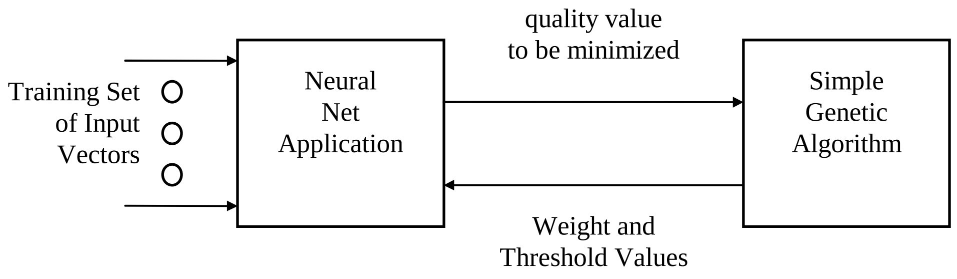


Figure 15. Neural Net and Genetic Algorithm Interface

A sufficient number of runs was established that created an extensive set of weight and threshold values. The SGA was executed with population sizes that ranged from 50 to 100, the chromosome length was set at 90, maximum number of generations ranged from 25 to 50, crossover probability ranged from 0.6 to 0.9, mutation probability was set to 0.01, and the random number seed ranged from 0.123 to 0.879.

In conclusion, most of the sets of weight and threshold values produced by the SGA enabled the neural net to have a better chance to discover the proper clustering of data since the gradient descent was already placed within the boundaries of global minima. The following table shows the extent of the explicit examination of data sets that contributed to the discovery (through the simulator behavior) of the optimization methods described earlier.

	XOR (4 vectors)	**Arbitrary (4 vectors)**	**Arbitrary (6 vectors)**
Number of Iterations Per Run	1000, 10000, 20000, 100000, 200000	1000, 10000, 20000, 100000, 200000	1000, 10000, 20000, 100000, 200000
Attract/Repel Method Distinct sets of initial conditions (weights and threshold values)	50	50	50
Median Method Distinct sets of initial conditions (weights and threshold values)	50	50	50
Temperature Adjustment Method Distinct sets of initial conditions (weights and threshold values)	100	100	100
Temperature & SGA Method Distinct sets of initial conditions (weights and threshold values)	50	50	50

Table 2 Further Refinement Runtime Cases

8. Conclusion

The Coulomb Energy network is a brilliant model that offers a unique perspective and approach to data transformation mappings and by extension to pattern classification. However, it has not caught on because of the problems in training it with the known algorithms. The training algorithms presented here allow this model to be practically useful and to harvest its power in applications requiring the kind of mappings that we have explained.

As closing remarks we would like to explain here in intuitive terms why the model has had difficulties with training; we consider this intuition into the model to be very interesting.

As explained, the network tries to "make" its outputs behave like charged particles in an electrostatic field. In simulating a reduction of the potential energy it is expected that the network's outputs will simulate exactly the position vectors of the charged particles which move freely around under only the constraints of their mutual interactions. (A side comment here: for the purposes of this model, we may change the sign in the original Coulomb force equation so as to make similar charges attract and dissimilar repel each other; this is ok as it does not change the mathematics of the Coulomb field other than the signs, the rest of the math remain the same so we can still use the potential energy optimization as the guide to training the net.)

The problem arises with the assumption that the net can actually simulate the movement of the charged particles. This in reality is not quite that easy. Under the forces of the mutual interactions, the physical particles can move *independently* from one another. By this we mean that each particle is free to follow precisely the total sum force that it receives by the electrostatic interactions and there is no other obstacle to its movement. But this is not what happens in the net that tries to simulate this movement. The output vectors "move" around as a result of changing the weights. The weights are the only parameters that the training algorithm controls. When a weight value changes then the output vectors change; we then say that the output vectors have "moved" around. But with a change in a weight, generally all of the outputs move. So the movement of each output is not exactly independent from the movements of other outputs like it happens with the charged particles.

Now lets consider a snapshot of the net during training. Consider the output vectors which in the next training step will change to new ones as a result of adjusting the weights. Suppose we have charged particles at the exact positions indicated by the current net outputs. Let's assume for reference that a charge q_i is in the position indicated by the output vector Y_i and that charge q_j in the position indicated by output vector Y_j. Now lets assume that q_i receives such a force that pushes it eastbound and q_j one that pushes it westbound. Well, that is exactly how each one will move; there is nothing else to constraint their movement. Now the nets' training algorithm tries to change the weights so that Y_i and Y_j will follow these directions precisely. But there may not be a combination of changes in the weights that would allow Y_i to follow q_i's eastbound move *and at the same time* also allow Y_j to follow q_j's westbound move. These are the dynamics that may arise and which would put the training algorithm in a dynamic equilibrium that is manifested as a trap in local minima (and so do not allow the training algorithm to simulate the process precisely).

Apart from this trouble, we know that the 2-layer network can actually position the output vectors just about anywhere; we know that a two layer network can in principle approximate any mapping [5](given sufficient nonlinearity). This is the reason we went after a 2-layer network and new algorithm as opposed to the original approach of C. Scofield which allowed adding one layer at a time, each trained independently from the others. However, by simply going after the gradient decent of the potential energy the net cannot produce the configurations sequence (weight vectors sequence) that would make the outputs simulate faithfully the particles movements. There exist weight vectors to place the outputs in just about any arrangement; however, there may be no smooth transition from one to the other; in other words, the proximity of two different output vector arrangements may not translate to a proportional proximity of the weight values configuration and this is precisely why the gradient decent gets stuck.

So we know that it is possible to find a sequence of configurations to simulate the particle movement, but two configurations that are successive in this sequence may not be close enough (in the Euclidian sense) to allow a gradient decent to find them. It should then be obvious that to go from one configuration of weight values to a new one in a way that allows simulation of the Coulomb field, we have to look elsewhere or at least we know that we cannot rely on the gradient decent alone. The solution to this problem then is one of the other methods that are based on an efficient virtual search of the weight space. We took up the genetic algorithms approach. An alternative way to go is the simulated annealing. Both of them worked just fine in our experimentation and they seem to be the most robust. The basic gradient decent method is prone to getting stuck in local minima as already explained, and so it should be used along with the tricks for escaping local minima (such as momentum terms, etc).

Acknowledgements

This work has been sponsored in part by NSF, and by DoD-Epscor and the Louisiana Board of Regents under grants: NSF/LEQSF (1992-93)-ADP-04 and F49620-98-1-0351.

References

[1] Scofield C., “Learning Internal Representations in the Coulomb Energy Network”, in Proc. IEEE Inter. Conf. on Neural Networks (ICNN ‘88), New York:IEEE, July 1988.

[2] Srikanth, R., and Koutsougeras, C., "Pattern Classification Using the Hybrid Coulomb Energy Network", Proceedings of the Fifth Annual Conference on Neural Nets and PDP, April 1992.

[3] Johnson, C.R., “Computational Inverse Problems in Medicine”, Computational Science and Engineering, IEEE Computer Society, Winter 1995, pp. 42-45

[4] Rumelhart D. E., Hinton G. E., and Williams R. J., “Learning Internal Representations by Error Propagation”, in “Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations”, editors Rumelhart David E., McClelland James L., and The PDP Research Group, MIT Press, 1986, pp. 151-193.

[5] Funahashi, K.I., "On the Approximate Realization of Continuous Mappings by Neural Networks", Neural Networks, Vol.2, 1989.

[6] Kohonen, T., “Representation of sensory information in self-organizing feature maps, and relation of these maps to distributed memory networks”, SPIE Vol.634, Optical and Hybrid Computing, 1986, pp. 248-259. Also, Kohonen, T., “Self-Organization and Associative Memory”, volume 8 of Springer Verlag Series in Information Sciences, Springer verlag, New York, 1984.

[7] Erkki Oja, and Kohonen T., “The Subspace Learning Algorithm as a Formalism for Pattern Recognition and Neural Networks”, Pankaj Mehra and Benjamin W. Wah, “Artificial Neural Networks: Concepts and Theory”, IEEE Computer Society Press, 1992, pp. 101-108.

[8] Caudill M., and Butler C., “Naturally Intelligent Systems”, MIT Press, 1993, pp. 37-109

[9] Rumelhart David E., and Zipser D., “Feature Discovery by Competitive Learning”, in “ Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations”, editors Rumelhart David E., McClelland James L., and The PDP Research Group, MIT Press, 1986, pp. 151-193.

[10] Smith, Robert E., Goldberg, David E., and Earickson, Jeff A., “SGA-C: A C-language Implementation of a Simple Genetic Algorithm”, TCGA Report No. 91002, March 2, 1994.