Genetic Synthesis of Unsupervised Learning Algorithms

Ali DAŞDAN and Kemal OFLAZER

Department of Computer Engineering and Information Science
Bilkent University
06533 Bilkent, Ankara, TURKEY
Email : dasdan@bcc.bilkent.edu.tr

Abstract

This paper presents new unsupervised learning algorithms that have been synthesized using a genetic approach. A set of such learning algorithms has been compared with the classical Kohonen's Algorithm on the Self-Organizing Map and has been found to provide a better performance measure. This study indicates that there exist many unsupervised learning algorithms that lead to an organization similar to that of Kohonen's Algorithm, and that genetic algorithms can be used to search for optimal algorithms and optimal architectures for the unsupervised learning.

1 Introduction

Genetic algorithms (GAs) [2, 4] are search and optimization algorithms which emulate the mechanics of natural evolution. GAs operate on a population of individuals, each of which consists of a string, called a *chromosome*, that encodes a solution to the problem being solved. Each individual is used to explore a different region of the search space of the problem. At each iteration, called a *generation*, a new population is created using probabilistic rules to exchange the information in the chromosomes of the individuals. In this way, GAs both exploit the information already gained and explore new regions in the search space.

The design of artificial neural networks (ANNs) whose performance is optimized for a certain application is still a research issue [8]. This design problem is also a complicated one because of the large design space, the presence of many variables, and complex interactions among these variables [5]. GAs can be used to search for optimal ANNs. A number of researchers have already applied GAs to synthesize application-specific ANNs [8]. Some have used GAs to determine the connection weights in an ANN, while others have used them to find the optimal topology and optimal parameters of a certain learning rule for an ANN. Chalmers [1] has used a GA to find better learning rules for a single layer ANN. Up to now, all the applications have been done in ANNs employing a supervised learning process.

In this work, we apply GAs to find better learning rules for Self-Organizing Map (SOM) [9, 10]. The SOM is an ANN with a two-dimensional array of output nodes and one-dimensional array of input nodes, which employs an unsupervised learning algorithm known as *Kohonen's algorithm*. Kohonen's algorithm uses a learning or weight update rule called *Kohonen's rule* to update the connections weights in the SOM architecture.

Kohonen [11] raises some questions about the self-organization process, which have remained unanswered so far. Three of them are,

1. Do there exist several, possibly many, optimal algorithms that lead to a similar organization produced by Kohonen's algorithm?
2. Does the basic recursive map algorithm (Kohonen's algorithm) ensue from some more general principle? If such a principle exists, does it possibly define in this category an optimal recursive algorithm that is more effective than the ones used so far?
3. Can the principle also be expressed for a more general network structure?

In this study, we have used the SOM architecture. We have encoded the learning or weight update rules as a chromosome string. We have then used a GA to find unsupervised learning rules with better performance based on a certain performance measure. We have found a number of such learning rules which perform better than Kohonen's rule for the applications we have investigated. Our study indicates that there are many algorithms leading to an organization similar to the one produced by Kohonen's algorithm for at least certain applications, and that the use of GAs can help us to answer the second and third questions raised by Kohonen.

The organization of the paper is as follows : We present an overview of genetic algorithms and applications of genetic algorithms to the design of ANNs in Section 2. Section 3 is devoted to the explanation of the experimental framework, in which Kohonen's Algorithm, the genetic algorithm, the genetic representation methods, and performance criteria are given. The experiments and results are discussed in Section 4. Finally, Section 5 includes the conclusions.

2 Genetic Algorithms and Connectionism

2.1 Genetic Algorithms

Genetic Algorithms [2, 4] are essentially search and optimization algorithms which work by mimicking the process of natural evolution as a means of advancing toward the optimum. They can search large and complex spaces effectively. GAs are robust in that they adapt to a wide variety of environments.

GAs use a population of individuals, each of which consists of a chromosome that encodes a solution to the problem for which GAs are employed. GAs are blind search algorithms in that they usually do not know what they are solving [4]. The connection between a GA and a problem is provided with an *evaluation function*. The evaluation function is calculated with an *evaluation algorithm*. The evaluation function measures how good an individual is in adapting to its environment, that is, it indicates the goodness of an individual, called its *fitness*, which is determined by examining, according to the requirements of the problem, the goodness of the solution encoded in the individual's chromosome.

A typical GA works as follows: First, the chromosomes in the population are initialized uniformly at random. At each generation, the individuals in the current population are evaluated using the evaluation function. Based on fitness values, individuals are selected from the population, two at a time, as *parents*. The individuals with better fitness values have a higher probability of being selected. A number of genetic operators are next applied to the parents. The most common genetic operators are *crossover*, and *mutation*. Using crossover, randomly selected portions of parents are exchanged, and two *children* are generated. These children are then mutated by producing random changes in their chromosomes. After that, the children are evaluated and a new generation is formed by selecting some of the parents and children on the basis of their fitness values in order to keep the population size constant.

Crossover and mutation are not applied to every pair of parents selected in a generation. The number of crossover operations is controlled with a probability called *the probability of crossover* and the number of mutation operations with a probability called *the probability of mutation*. The crossover operator is responsible, by generating children similar to their parents, for the exploitation of knowledge already gained, and responsible, by producing new individuals, for the exploration of the solution space. The mutation operator is used to increase the diversity of the population. The probability of crossover and the probability of mutation should be selected so that the balance between the exploitation of good solutions found so far and the exploration of the solution space is maintained. The probability of crossover is usually very near to 1 and the probability of mutation is near to 0.

2.2 Genetic Connectionism

The design of an artificial neural network which is optimized for a specific application is still a research issue [5, 8]. The space of possible ANN architectures and learning rules is extremely large. There are many variables, both discrete and continuous, and they interact in a complex manner [5]. Besides, the performance of an architecture and a learning rule usually depends on the application. So far, various heuristic methods for this design problem have been derived from experience in both practical and small applications.

Instead of extensive trial-and-error experimentation, the optimal architectures and learning rules for a certain application can be searched with a powerful search and optimization method. GAs are good candidates of such methods.

Genetic connectionism combines the genetic search and the connectionist computation. Up to now, GAs have been applied successfully to the problem of designing ANNs with supervised learning process in several ways [5, 8]. These are:

- Given the topology, (i.e., the connectivity, the number of layers, and the number of nodes within each layer), and the learning rule of the network, GAs are used to determine the connection weights. That is, the GA learning weights is compared to other learning algorithms, e.g., backpropagation.
- Given the learning rule of the network, GAs are used to determine the topology of the network.
- Given the topology of the network, GAs are used to find fittest learning rules based on a certain fitness measure [1].
- Given the topology and the learning rule of the network, GAs are used to determine optimal parameters for the learning rule.

2.3 Related Work

Our work is in the same category with that of Chalmers [1] in the sense that we both use a GA to find fittest learning rules but fixing the network topology in advance. However, Chalmers uses an ANN with a supervised learning process but we use an ANN, namely SOM, with an unsupervised learning process.

Chalmers encodes the learning rule of a supervised learning algorithm in a fully connected, single layer, feed-forward neural network with sigmoid output units. The fitness of a learning rule is determined by applying it to a number of learnable tasks. Each task is a linearly separable mapping from input patterns to output patterns. A genetic algorithm is employed to the find the fittest learning rules in this neural network. Many fittest learning rules are found at the end of the experiments. Based on the fitness measure on the tasks, the performance of the learning rules on the tasks is comparable to that of the well-known delta rule.

3 Experimental Framework

3.1 The Self-Organizing Map

The Self-Organizing Map (SOM) [9, 10, 11, 12] is an ANN in which the cells become specifically tuned to various input signal patterns or classes of patterns through an unsupervised learning process. The self-organizing process can be considered as a mapping of the probability density function of the high dimensional input data onto the two-dimensional display, where output nodes corresponding to nearby input patterns lie topologically nearby. The aim is to cluster the input data. Similar inputs should be classified as being in the same cluster.

In the SOM architecture, there are n continuous-valued inputs x_1 to x_n, defining a point $\vec{x}$ in n-dimensional real space $\mathcal{R}^n$. $\vec{x}$ is called the *input vector* or the *input pattern*. There are k output cells and they are arranged in a 2-dimensional array. Each output cell o_i, $i \in \{1, \cdots, k\}$, is connected to x_j, $j \in \{1, \cdots, n\}$, with a weight w_{ij}. The weight vector $\vec{m}_i$ of an output cell o_i is an n-dimensional vector of weights, i.e., $\vec{m}_i = [w_{i1}, \cdots, w_{in}]^T \in \mathcal{R}^n$.

The measure for the match of $\vec{x}$ with $\vec{m}_i$ is based on the Euclidean distance $\| \vec{x} - \vec{m}_i \|$ of $\vec{x}$ and $\vec{m}_i$ defined as

$$\| \vec{x} - \vec{m}_i \| = \sqrt{\sum_{j=1}^{n} (x_j - w_{ij})^2} \tag{1}$$

The vectors $\vec{x}(t)$and $\vec{m}_i(t)$ represent the values of vectors $\vec{x}$ and $\vec{m}_i$ respectively at discrete time steps $t = 1, 2, \cdots$.

Kohonen's algorithm [9, 10], presented in Figure 1, proceeds as follows: First, all the weights from n inputs to the k outputs are initialized to small random values, usually between 0.4 and 0.6 [3]. The weights coming to a given output cell from all the input cells are normalized so that the total length of each weight vector from input cells to a given output is 1 [3]. After that, the network is trained T_{max} times. During each training iteration, a new input pattern $\vec{x}$ selected randomly from all the patterns is presented to the network. The Euclidean distances between the input vector $\vec{x}$ and the output vectors $\vec{m}_i$ are then computed using the Eq. 1. The *best matching cell* o_c, the one with the minimum Euclidean distance to the input pattern $\vec{x}$, is identified. That is,

$$\| \vec{x}(t) - \vec{m}_c(t) \| = \min_{1 \le i \le k} \| \vec{x}(t) - \vec{m}_i(t) \| \tag{2}$$

We refer to the best matching cell as the *winner*. Next, the weights of all the cells within a *neighborhood* $N_c(t)$ of the winner o_c are updated using the equation

$$\vec{m}_i(t+1) = \begin{cases} \vec{m}_i(t) + \alpha(t)[\vec{x}(t) - \vec{m}_i(t)] & \text{if } i \in N_c(t) \\ \vec{m}_i(t) & \text{otherwise} \end{cases} \tag{3}$$

where $\alpha(t)$, the learning rate, is a scalar-valued function such that $0 < \alpha(t) < 1$. The neighborhood $N_c(t)$ at a time step t of the the winner o_c consists of all the cells lying in a small area around the cell o_c at the time step t. The width or radius $r(t)$ of the neighborhood and the learning rate decrease monotonically with the time step t. The learning rate remains constant within the neighborhood. The exact forms of $\alpha(t)$ and $r(t)$ are given in Figure 1. After training the network, the calibration step comes. During this step, each pattern is presented to the network once and the winner for the pattern is labeled with the item (or symbol) representing the pattern.

The ordering of the $\vec{m}_i$ occurs during the first $T = 1000$ training iterations. Thus, T_{max} should be larger than 1000. The remaining training iterations are only needed for the fine adjustment of the map. Moreover, the time complexity of Kohonen's Algorithm is directly proportional to T_{max}. Hence, we have selected a small value for T_{max}. T_{max} is equal to $30 \times k$, where k is the number of output cells or the size of the map and $k = 10 \times 7 = 70$. Hence, $T_{max} = 2100$.

3.2 Genetic Representation

A general weight update rule for the SOM should incorporate relevant variables and subsume Kohonen's rule (Eq. 3). These variables consist of the inputs, the outputs, the connection weights, and the training iteration number (or the time step).

Let $\vec{x}$ be the input vector and o_i be the i^{th} output cell, where $i \in \{1, \cdots, k\}$. There are n connections between o_i and input nodes $x_1, x_2, \cdots, x_n$. The weight w_{ij} of the connection between o_i and x_j is increased at the time step t by an amount equal to Δw_{ij} where

$$\Delta w_{ij} = \mathcal{F}(w_{ij}, x_j, t, y_i) \tag{4}$$

where y_i depends on the correlation between $\vec{x}$ and $\vec{m}_i$.

When calculating the value of the function $\mathcal{F}$ at a time step t, we use the values of the variables of the function $\mathcal{F}$ at the time step t. The function $\mathcal{F}$ is defined as

$$\begin{aligned} \mathcal{F}(w_{ij}, x_j, t, y_i) = \quad c_0 \quad & (c_1 w_{ij} + c_2 x_j + c_3 t^{-1} + c_4 y_i + c_5 w_{ij} x_j + c_6 w_{ij} t^{-1} + \\ & c_7 w_{ij} y_i + c_8 x_j t^{-1} + c_9 x_j y_i + c_{10} t^{-1} y_i + c_{11} w_{ij} x_j t^{-1} + \\ & c_{12} w_{ij} x_j y_i + c_{13} w_{ij} t^{-1} y_i + c_{14} x_j t^{-1} y_i + c_{15} w_{ij} x_j t^{-1} y_i) \end{aligned} \tag{5}$$

where $t > 0$ and the coefficients $c_0, \cdots, c_{15}$ are signed real numbers.

When the coefficients other than c_0, c_1, and c_9 are 0, the function $\mathcal{F}$ gives a version of Kohonen's rule. Namely, $\mathcal{F}$ becomes

$$\begin{aligned} \mathcal{F}(w_{ij}, x_j, t, y_i) = & \quad c_0(c_6 w_{ij} t^{-1} + c_8 x_j t^{-1}) \\ = & \quad c_0 t^{-1}(c_0 c_8 x_j + c_0 c_6 w_{ij}) \end{aligned} \tag{6}$$

Algorithm: Kohonen's Algorithm
Input: input patterns.
Output: a clustering of patterns on a map.

1. read input patterns
2. normalize input patterns, i.e., map each element into the interval [0, 1].
3. assign a random weight in $[0.4, 0.6]$ to every connection weight
4. normalize weights so that the total length of the each weight vector from all the inputs to a given unit is 1, i.e., for each $i \in \{1, \cdots, k\}$
$$\vec{m}_i \leftarrow \vec{m}_i / \parallel \vec{m}_i \parallel$$
5. for each time step t in $[1, 2100]$ do /* training the network $T_{max} = 2100$ times*/
   - 5.1. get a pattern randomly
   - 5.2. find the winner for the pattern
   - 5.3. find $\alpha(t)$ and radius $r(t)$. The $\alpha(t)$ is defined as
$$\alpha(t) = \begin{cases} \alpha(0) + (t/T)(\alpha(T) - \alpha(0)) & \text{if } t \leq T \\ \alpha(T) + ((t - T)/(T_{max} - T))(-\alpha(T)) & \text{otherwise} \end{cases}$$
     where $T = 1000, \alpha(0) = 0.5, \alpha(T) = 0.04, T_{max} = 2100$, and the radius $r(t)$ of the neighborhood of the winner at the time step t is defined as
$$r(t) = \begin{cases} r(0) + (t/T)(r(T) - r(0)) & \text{if } t \leq T \\ 1 & \text{otherwise} \end{cases}$$
     where $T = 1000, r(0) = 6, r(T) = 1$,
   - 5.4. update weights of the cells in the neighborhood of the winner,
6. endfor
7. for each pattern do /* calibration */
   - 6.1. find the winner for the pattern,
   - 6.2. assign the symbol of the pattern to the winning cell,
8. endfor

Figure 1: Kohonen's Algorithm

which is equivalent to Eq. 3 for $c_8 = 1, c_6 = -1$, and $\alpha(t) = c_0 t^{-1}$.

We have encoded these coefficients in a number of different ways:

3.2.1 Coding 1

Each coefficient is coded using 5 bits. This coding is similar to Chalmers' [1]. The interpretation of these 5-bit strings corresponding to each coefficient is as follows:

The coding of the coefficient c_0 is different from that of the other coefficients. Let c_0 be coded using the bits $b_1b_2b_3b_4b_5$, and d_0 be the decimal value of the bits $b_2b_3b_4b_5$ encoding the coefficient c_0. Then,

$$c_0 = \begin{cases} 0 & \text{if } d_0 = 0 \\ -2^{d_0-15} & \text{if } d_0 \neq 0 \text{ and } b_1 = 0 \\ 2^{d_0-15} & \text{if } d_0 \neq 0 \text{ and } b_1 \neq 0 \end{cases} \tag{7}$$

Thus, the value of c_0 is such that $-2^{-14} \leq c_0 \leq 1$. Let c_j, where $j \in \{1, \cdots, 15\}$, be coded using the bits $b_1b_2b_3b_4b_5$, and d_j be the decimal value of the bits $b_2b_3b_4b_5$. Then,

$$c_j = \begin{cases} 0 & \text{if } d_j = 0 \\ -2^{d_j-1} & \text{if } d_j \neq 0 \text{ and } b_1 = 0 \\ 2^{d_j-1} & \text{if } d_j \neq 0 \text{ and } b_1 \neq 0 \end{cases} \tag{8}$$

Thus, the value of c_j is such that $-1 \leq c_j \leq 2^{14}$.

Each coefficient c_0c_j, $j \in \{1, \cdots, 15\}$, can take values between between -2^{-14} and 2^{14} though not exhaustively. One can see that the 80-bit chromosome corresponding to $\mathcal{F}$ can generate approximately 10^{24} weight update rules.

For this coding, y_i is the correlation between $\vec{x}$ and $\vec{m}_i$, mapped onto the interval $(0, 1)$ using a sigmoid function. The formula for y_i is given as

$$y_i = \frac{1}{1 + e^{-y_i'}} \tag{9}$$

where

$$y_i' = \sum_{j=1}^{n} x_j w_{ij} \tag{10}$$

3.2.2 Coding 2

The coefficient c_0 is coded in 5 bits and the others in 8 bits. The coding of the coefficient c_0 is exactly similar to that in coding 1. The coding of the other coefficients is different. Let c_j, $j \in \{1, \cdots, 15\}$, be coded using the bits $b_1b_2b_3 \cdots b_8$, and d_j be the decimal value of the bits $b_2b_3 \cdots b_8$. Then,

$$c_j = \begin{cases} -2d_j & \text{if } b_1 = 0 \\ 2d_j & \text{if } b_1 = 1 \end{cases} \tag{11}$$

Thus, the value of c_j is such that $-2(2^7 - 1) \leq c_j \leq 2(2^7 - 1)$. The 125-bit chromosome can generate approximately 10^{38} weight update rules. For this coding, y_i is the correlation between $\vec{x}$ and $\vec{m}_i$. The formula for y_i is given as

$$y_i = \sum_{j=1}^{n} x_j w_{ij} \tag{12}$$

3.2.3 Coding 3

This coding is the same as coding 2 except that the coefficients other than the coefficient c_0 are calculated differently. Let c_j, $j \in \{1, \cdots, 15\}$, be coded using the bits $b_1 b_2 b_3 \cdots b_8$, and d_j be the decimal value of the bits $b_2 b_3 \cdots b_8$. Then,

$$c_j = \begin{cases} -2d_j/M & \text{if } b_1 = 0 \\ 2d_j/M & \text{if } b_1 = 1 \end{cases} \tag{13}$$

where $M = 2^7 - 1$, the maximum integer that fits in 7 bits. Thus, the value of c_j is such that $-2 \leq c_j \leq 2$. The 125-bit chromosome can generate approximately 10^{38} weight update rules.

3.2.4 Coding 4

This coding is the same as coding 3 except that the formula for y_i is different. For this coding, y_i is the correlation between $\vec{x}$ and $\vec{m}_i$, mapped onto the interval $(0, 1)$ using a sigmoid function. The formula for y_i is given as

$$y_i = \frac{1}{1 + e^{-y_i'}} \tag{14}$$

where

$$y_i' = \sum_{j=1}^{n} x_j w_{ij} \tag{15}$$

The 125-bit chromosome can generate approximately 10^{38} weight update rules.

The length of the interval between any two successive values that a coefficient other than c_0 can take is not constant in coding 1 but constant in coding 2, 3, and 4. The coefficient c_0 limits the values of other coefficients. The value of c_0 is almost always less than 1 because the coefficients should be small numbers. When we removed the coefficient c_0 from any coding, the weight update rules obtained performed very badly. That is, they had no self-organizing power at all. Thus, the coefficient c_0 is essential.

3.3 The Genetic Algorithm

We have performed our experiments using a program implementing the genetic algorithm in Figure 2. The properties of the genetic algorithm we used are as follows:

- the number of individuals in the population is 40, and the population size is constant,
- the maximum number of generations is 400,
- the probability of crossover is 0.9,
- the probability of mutation is 0.01,
- Roulette Wheel selection method [4] is used,
- elitism [1, 4], which is including the fittest individual in a generation in the next generation, is used.

The genetic algorithm works as follows: First, all the pattern sets are read. The number of the pattern sets is 4, each set corresponding to a sample problem. The pattern sets are next normalized as in Kohonen's Algorithm. The chromosomes in the population are initialized with a fair coin toss. That is, each bit of a chromosome is set to 1 or 0 with a probability of 0.5. During the entire run of the program, each pattern set is used an equal number of times to evaluate the rules, but the order of selection of the sets is random. For example, during 400 generations, each set of patterns is selected 2 times and the rules are evaluated 50 times on each set.

The main loop of the algorithm contains the steps corresponding to the evaluation of the population, calculation of the statistics about the population, and creation of a new generation. The individuals in the population are evaluated using the Evaluation Algorithm, which is given in section 3.5. The statistics about the population consists of the maximum fitness value, the individual with the maximum fitness, and so on. In order to create a new generation, all the individuals are selected from the current population, two at a time, as parents. Selection is based on the fitness values of the individuals so that individuals with higher fitness values have a higher chance of being selected. The genetic operator crossover is next applied to the parents. Using crossover, randomly selected portions of parents are exchanged, and two children are produced. These children are then mutated by producing random changes in their chromosomes. The number of crossover operations is controlled with the probability of crossover, and the number of mutation operations with the probability of mutation.

Since the time complexity of the genetic algorithm is directly proportional to the number of generations, the population size, and the time complexity of the evaluation algorithm, we kept the population size and the maximum number of generations rather small. Using greater values for these variables caused a very large increase in the runtime of the genetic algorithm.

3.4 Fitness Calculation

In the SOM architecture, the definition of the optimal mapping is still unclear [12]. The mean E, which we call the mean error, of the Euclidean distances between input patterns and the weight vectors of their best matching output cells is used as a criterion for the optimal mapping [12]. Formally, let o_{cp}, where $c \in \{1, \cdots, k\}$ and $p \in \{1, \cdots, P\}$ with P being the number of patterns, be the best matching cell for a pattern $\vec{x}_p$ during calibration. Then, given the weight vector $\vec{m}_{cp}$ of an output cell o_{cp}, the Euclidean distance between $\vec{x}_p$ and $\vec{m}_{cp}$ is as given in Eq. 1. We define the mean error E for all P patterns as

$$E = (1/P) \sum_{p=1}^{P} \parallel \vec{x}_p - \vec{m}_{cp} \parallel \qquad (16)$$

According to this cost formulation, a map with the smallest E value is the best map. Since the best map should be the map with the highest fitness, we transform the smallest error to the highest fitness value in two different ways:

1. **Fitness criterion 1 :**
   Given a map with a mean error E_r, the fitness $F(E_r)$ of the weight update rule r

Algorithm: The Genetic Algorithm
Input: sets of input patterns.
Output: unsupervised learning rules.

1. read all the pattern sets
2. normalize the pattern sets as in Kohonen's Algorithm
3. initialize the population
4. for 8 iterations do /* A total of 400 generations since 8 * 50 = 400 */
   4.1. select a pattern set randomly among 4 pattern sets
   4.2. for 50 generations do
      4.2.1. for each individual in the population do
         4.2.1.1. evaluate the individual using the Evaluation Algorithm
      4.2.2. endfor
      4.2.3. calculate the statistics about the population
      4.2.4. for each pair of individuals in population do /* creating a new generation */
         4.2.4.1. select a pair of individuals as parents
         4.2.4.2. perform a two-point crossover on the parents with the probability of crossover, and so obtain two children
         4.2.4.3. mutate each bit of each children with the probability of mutation,
      4.2.5. endfor
   4.3. endfor
5. endfor

Figure 2: The Genetic Algorithm.

producing this map is defined as

$$F(E_r) = \begin{cases} \infty & \text{if } E_r = 0 \\ \frac{100}{E_r} & \text{if } E_r \neq 0 \end{cases} \tag{17}$$

where ∞ is a very large value.

2. **Fitness criterion 2 :**
Given a map with a mean error E_r, the fitness $F(E_r)$ of the learning rule producing this map is defined as

$$F(E_r) = max_{allr}(E_r) - E_r \tag{18}$$

where $max_{allr}(E_r)$ is the maximum mean error value among those of all the maps.

3.5 The Evaluation Algorithm

The evaluation algorithm, presented in Figure 3, calculates the fitness values of the individuals in the population based on one of the fitness criteria. The evaluation algorithm is very similar to Kohonen's algorithm. The most important difference is that we use the encoded weight update rule (Eq. 5) instead of Kohonen's rule (Eq. 3). The time complexity of the evaluation algorithm is exactly the same as that of Kohonen's Algorithm.

Algorithm: The Evaluation Algorithm.
Input: an individual in the population.
Output: the individual whose fitness is computed.

1. decode the coefficients of $\mathcal{F}$ from the chromosome of the given individual
2. initialize the map as in Kohonen's Algorithm (steps 3 and 4)
3. for each time step t in $[1, 2100]$ do /* training the network $T_{max} = 2100$ times */
   3.1. get a pattern randomly
   3.2. find the best matching cell
   3.3. find the radius $r(t)$ as in Kohonen's Algorithm (step 5.3)
   3.4. update weights of the cells in the neighborhood of the winner using the rule decoded from the chromosome, (i.e. use the function $\mathcal{F}$)
4. endfor
5. for each pattern do /* calibration */
   5.1. find the winner for the pattern
6. endfor
7. find the fitness of the individual using the fitness criterion

Figure 3: The Evaluation Algorithm

3.6 Sample Problems

We have used six sample problems for experimentation. The first sample problem "Taxonomy of Abstract Data" is from Kohonen [10]. It consists of abstract data vectors with hypothetical attributes. The attributes are determined by assuming hierarchical structures among the vectors. The second sample is obtained from McClelland *et al.* [13]. It consists of a number of nouns with certain features. Based on the features of the object implied by the noun, each noun is associated with a vector of features. The third sample problem is taken from Waltz *et al.* [14]. It is composed of a number of "concepts" and a number of "features". Each concept is associated with a vector of values where each value indicates the level of activation of the feature of the concept. The fourth sample problem includes a number of nouns. Each noun has a vector whose entries give the ordinal values of the letters of the noun. The fifth sample problem, given in Figure 10, contains 400 points uniformly distributed in a unit square. The sixth sample problem, given in Figure 11, contains 400 points uniformly distributed in a plus-shaped polygon inside a unit square. The first four problems were used as both training and test data. The last two problems were used only as test data. The problems are given in Appendix A.

4 Experiments and Results

The details of the runs are given in Table 1. We have experimented with different genetic codings and different fitness criteria. The learning rules in the population were trained on the first four sample problems. The genetic algorithm was run for 400 generations. For each

50 generations, a sample problem was selected to train the rules. Each sample problem was selected exactly 2 times but the order of the selection was random.

After the run of the genetic algorithm, a number of best learning rules were taken from the population and tested. All the selected rules and Kohonen's rule were tested on the first four sample problems (training data), which were used to train the rules. Two of the rules and Kohonen's rule were also tested on the last two sample problems (test data), which were not used to train the rules.

Kohonen's rule (Eq. 3) was tested on each sample problem 100 times, each time with a different random number seed. Our rules were also tested on each sample problem 100 times, but we used the same random number seeds as those used in testing Kohonen's rule in order to ensure that the same initial maps were used for all the rules.

We give 11 new unsupervised learning rules in Table 2 and Table 3. The entries in the tables correspond to the coefficients in the Eq. 5. The algorithm used to test our rules is very similar to Kohonen's Algorithm. The only difference is that we have used our rules (Table 2 and Table 3) instead of Kohonen's rule as the weight update rule.

The comparisons of the performance of our rules and Kohonen's rule on the first four sample problem are given in Table 4. The entries in the first column give the rule number. To explain other entries, consider the first rule, i.e., the first row. The mean error produced by the first rule is smaller than the one produced by Kohonen's rule in 70% of all the runs for the sample problem 1. Similarly, the mean error by the first rule is smaller than the one by Kohonen's rule in 99% of all the runs for the second sample problem. The interpretation of the other entries is similar.

The comparisons of the performance of the second rule, the fifth rule, and Kohonen's rule on the last two sample problems are given in Table 5. The interpretation of the entries is similar to that of the entries in Table 4.

In Figure 4 and Figure 5, we present some maps taken from the test runs of our first rule and Kohonen's rule on the training data. The original maps are all hexagonal but some are converted to rectangular maps. In Figure 6, Figure 7, Figure 8, and Figure 9, we present some maps taken from the test runs of our second rule and Kohonen's rule on the test data. We also give the mean errors obtained for these maps. In these maps, the points represent the positions of the output cells in the weight space, and a line is plotted between two points corresponding to two neighbor output cells. Moreover, a higher density of grid points in a region on these maps usually indicates a higher probability of the random input patterns in the region. We used the following parameters in the runs on the test data: initial learning rate = 0.8, final learning rate = 0.04, initial radius = 20, final radius = 1, and the number of training iterations = 80000.

We now give some observations:

- Learning rules obtained from Run #1 perform the best among all the other rules including Kohonen's rule.
- Considering the genetic coding criteria, we can rank the rules from the best to the worst in terms of their performance as: rules from coding 1, rules from coding 2, rules from coding 3, rules from coding 4.
- Rules obtained from runs other than Run #1 and Run #2 perform poorer than Kohonen's rule even on the data used in training.
- Considering the rule #2 taken from Run #1, we can say that the performance metric is good enough to make comparisons based on it.

Table 1: Classification of Experiments.

Run #	*Coding #*	*Fitness Crit. #*	*Rules*	*Input Data*
1	1	1	1, 2, 3	problems 1, 2, 4 are normalized, but not 3
2	2	2	4, 5	"
3	3	2	6, 7	"
4	4	1	8, 9	"
5	3	1	10, 11	problems 1, 2, 3, and 4 are normalized

Table 2: Coefficients of Six New Unsupervised Weight Update Rules.

c_i	*New Learning Rules*					
	1	2	3	4	5	6
c_0	2^{-13}	2^{-13}	2^{-14}	0.03	0.03	0.000488
c_1	0	0	2^{13}	−1.95	−1.95	−184
c_2	−2	−2	-2^{13}	1.94	1.94	244
c_3	2	2	0	0.00	0.00	0
c_4	2^9	2^{12}	2^{11}	−0.38	−0.38	−146
c_5	-2^5	2	2^3	−0.08	−0.08	−70
c_6	2^{12}	2^{12}	-2^3	−1.34	−1.34	−250
c_7	0	−2	-2^{14}	−0.27	−0.77	64
c_8	-2^{12}	-2^{12}	2^6	1.20	1.20	174
c_9	2^6	2^2	2	1.34	1.34	182
c_{10}	-2^6	2^6	2^{12}	−1.01	1.01	−240
c_{11}	−1	0	0	0.31	0.31	82
c_{12}	-2^{13}	2^5	2^{12}	1.73	1.73	22
c_{13}	2^9	2^9	-2^{14}	−0.31	−0.31	142
c_{14}	-2^3	-2^3	−2	0.17	0.17	−214
c_{15}	2^8	2^8	2^3	1.54	1.54	42

- Considering the rules (especially rule #2) taken from Run #1, we can say that there exist unsupervised learning rules that produce maps similar to those by Kohonen's rule and that perform better as least on the problems we have investigated.
- Considering the output maps produced, we can say that the clustering obtained by some of our rules and by Kohonen's rule are very similar.

5 Conclusions

We have presented an overview of genetic algorithms, and the use of genetic algorithms in the design of artificial neural network architectures. We have applied genetic algorithms to the design of unsupervised learning rules in the self-Organizing Map architecture. We have discovered a number of such rules. Some of these rules as well as Kohonen's rule have later been tested both on the training data and on the test data. Some of our rules have performed much better than Kohonen's rule on all the sample problems.

Table 3: Coefficients of Five New Unsupervised Weight Update Rules.

c_i	*New Learning Rules*				
	7	8	9	10	11
c_0	0.001953	0.0625	0.2500	0.06	0.06
c_1	−124	−0.6772	−0.6457	−0.90	−0.96
c_2	192	1.3543	1.3543	1.07	1.07
c_3	−4	0.0472	0.0000	−0.02	−0.02
c_4	250	−1.1969	1.1969	−0.93	−0.91
c_5	−174	0.1575	0.1260	−0.05	−0.02
c_6	−218	−1.0709	−1.2283	−1.51	−1.53
c_7	134	1.7953	1.2913	−0.61	−1.59
c_8	184	0.4252	0.4252	1.42	1.43
c_9	−126	−0.0945	−0.7402	−1.46	2.00
c_{10}	102	−1.5906	−1.5276	−0.17	0.05
c_{11}	108	−0.5512	−0.5197	−0.05	−0.05
c_{12}	24	−0.3150	−0.3150	−0.96	−0.77
c_{13}	−54	−0.0157	0.0157	0.17	1.43
c_{14}	10	0.9764	0.9764	0.28	1.42
c_{15}	−198	1.4488	1.2756	−0.65	−0.65

Table 4: Performance of Rules on the First Four Sample Problems.

Rule #	*On Sample 1*	*On Sample 2*	*On Sample 3*	*On Sample 4*
1	70	99	80	100
2	71	99	98	100
3	60	95	85	100
4	79	96	0	100
5	81	96	0	100
6	3	72	0	100
7	1	26	0	56
8	0	0	0	100
9	1	0	0	97
10	11	92	0	100
11	24	94	0	100

Table 5: Performance of Some Rules on the Last Two Sample Problems.

Rule #	*On Sample 5*	*On Sample 6*
2	80	100
5	0	0

```
For sample problem 1 : (Hexagonal)

   C B A * Z * Y * 5 6          R Q P * O * N * D C
  * D * * * * * * * *          * * * W * * * M E B
   E * * V U * X 2 3 4          * * * * * * * * F A
  * * F * * * * * 1 *          * 2 1 * X * S L K G
   * H G * T * W * * *          3 * * * * * * * * H
  * I * K S * * * Q *          * 4 * Y * * * T * I
   J * L M N O P * * R           5 6 * Z * V U * * J
        (a)                            (b)

For sample problem 2 : (Rectangular)

 bat      *  food  co-chicken  potato     *     woman   *   wolf  fl_bat
 chicken  *    *       *       pasta      *       *     *   lion    *
 ball     *  bb_bat  carrot    cheese   doll      *    man    *    dog
   *      *    *    hammer      *       *         *     *    boy    *
   *      *  spoon     *        *   li_chicken sheep  girl   *      *
 hatchet  *    *    plate       *       *         *     *     *    rock
 fork  paperwt vase    *       curtain  window  desk    *     *     *
                                 (a)

 bat      *       *      *     doll   *    woman   man    *     lion
 chicken  *     pasta  cheese    *    *       *     *    boy     dog
   *  co-chicken  *      *       *   desk     *    girl  wolf   fl_bat
   *   potato     *    carrot    *    *       *     *     *       *
 food     *       *      *       *  window    *     *    sheep  li_chicken
   *      *       *    hatchet  fork  *       *  curtain  *       *
 ball  bb_bat   hammer  spoon    *  paperwt  vase  plate  *      rock
                                 (b)
```

Figure 4: The maps obtained from test runs. The map (a) corresponds to the one produced by our first rule, and the map (b) to the one produced by Kohonen's rule.

```
For sample problem 3 : (Rectangular)

    * *     *       *     *    * casino   *       * weekend
    * *     *       *  fire-at *   *   gambling   *   *
    * *     *       *     *    *   *      *       * dollar
    * *     *       *     *    *   *  waste-money *   *
    * * video-games *     *    *   *      *       *   *
    * *     *       * outdoors *   *      *       *   *
    * *     *       *     *    *   *   hunting    *  deer
                                (a)

  dollar     *   *     *    *      *     *     *    * weekend
    *        *   *     * gambling  *     *     *    *    *
waste-money  *   *     *    *      *  fire-at  *    *    *
    *        *   *     *    *      *     *     *    *    *
    *        * casino  *    *      *     *     *    * hunting
    *        *   *     *    *  outdoors  *     *    *    *
video-games  *   *     *    *      *     *    deer  *    *
                                (b)

For sample problem 4 : (Rectangular)
   dog   *  curtain  *    *     *     *     *  wolf   spoon
    *  food    *     *    *   lion  plate  rock  *      *
    *  fork  doll   girl  *     *     *     *    *     sheep
   boy   *     *     *   desk   *  chicken  *  potato  woman
   vase  *    bat    *   ball   *     *     *    *      *
    *    *     *     *    *     *  cheese   *  window   *
  pasta  *    man    * hatchet  *  carrot   *    *     hammer
                                (a)

   vase  *    dog    *      *    *  hatchet  *  potato  window
   man   *   food    *      * chicken  *     *    *       *
   bat   *    *    boy      *    *     *     *    *     woman
    *   desk  *      *      *    *  cheese   *  sheep     *
   ball girl fork  doll     *    *     *     *    *     spoon
    *    *   lion    *      * hammer   *     *    *      wolf
  pasta  *    *   curtain   * carrot   *   plate  *      rock
                                (b)
```

Figure 5: The maps obtained from test runs. The map (a) corresponds to the one produced by our first rule, and the map (b) to the one produced by Kohonen's rule.

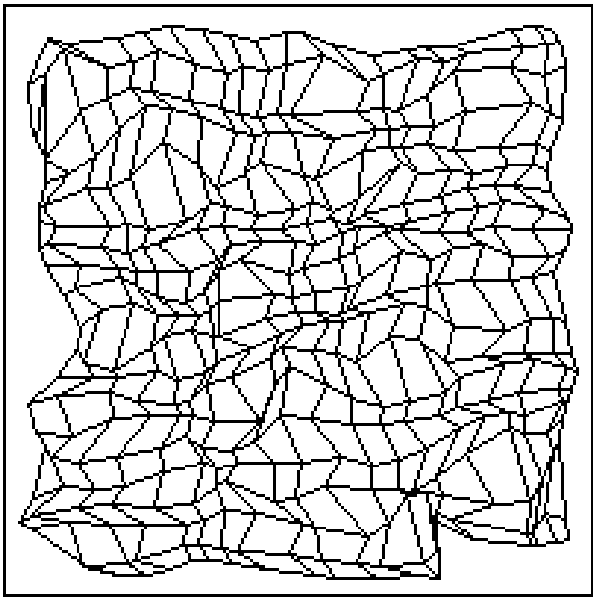

Figure 6: Map by Kohonen's Rule on Sample Problem 5 (Mean error = 0.055368).

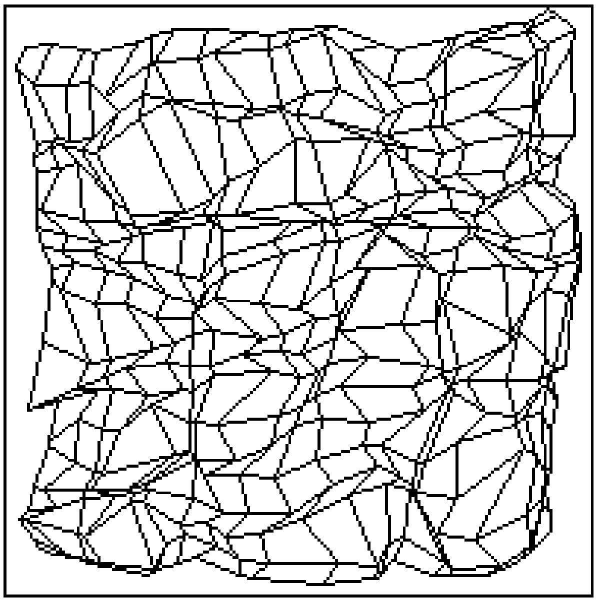

Figure 7: Map by Rule 2 on Sample Problem 5 (Mean error = 0.044887).

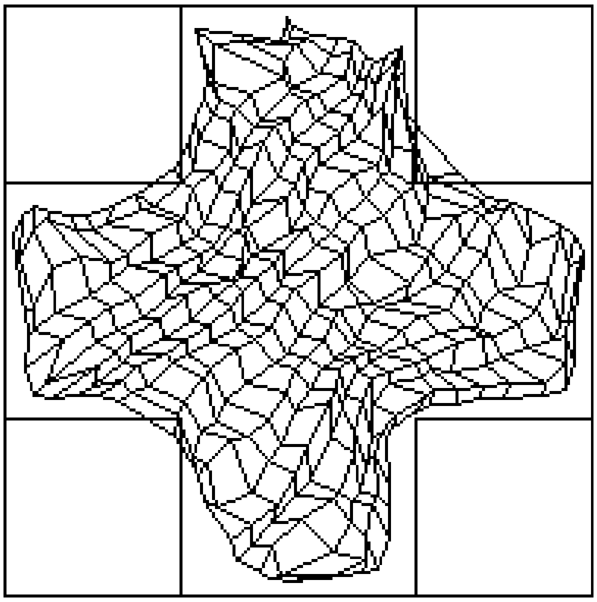

Figure 8: Map by Kohonen's Rule on Sample Problem 6 (Mean error = 0.051125).

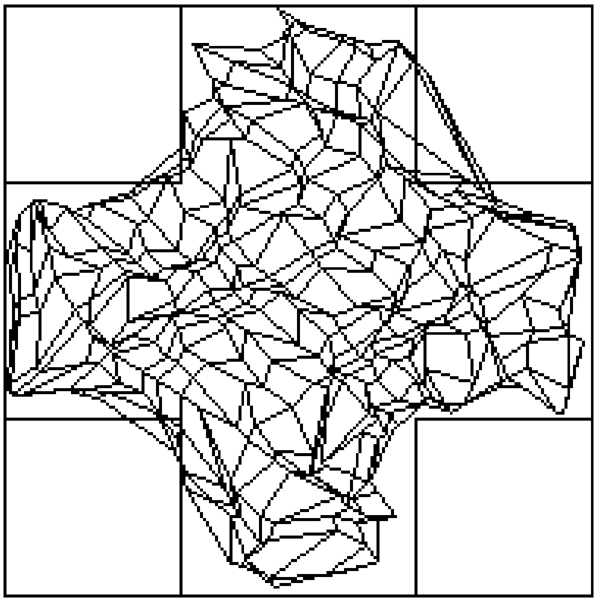

Figure 9: Map by Rule 2 on Sample Problem 6 (Mean error = 0.037750).

This study indicates that there are many unsupervised learning algorithms that lead to a similar organization like the one produced by Kohonen's algorithm. The ideas in this study can be used to obtain application-specific unsupervised learning algorithms. In addition, the examination of the resulting rules can help us find the general unsupervised learning algorithm, if one exists, applicable to a wide variety of artificial neural networks.

References

[1] Chalmers, D.J. (1990). The Evolution of Learning : An experiment in Genetic Connectionism. In Proc. of the 1990 Connectionist Models Summer School, eds. D.S. Touretzky, J.L. Elman, T.J. Sejnowski, and G.E. Hinton, San Mateo, CA : Morgan Kaufmann.

[2] Davis, L. (1991). What is a Genetic Algorithm? In Handbook of Genetic Algorithms, ed. L. Davis, pp. 1-22, New York : Van Nostrand Reinhold.

[3] Eberhart, R.L. and R.W. Dobbins (1990). Implementations. In Neural Network PC Tools, Academic Press.

[4] Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning, Reading : Addison-Wesley.

[5] Harp, S.A. and T. Samad (1991). Genetic Synthesis of Neural Network Architectures. In Handbook of Genetic Algorithms, ed. L. Davis, pp. 202-221, New York : Van Nostrand Reinhold.

[6] Hertz, J., A. Krogh and R.G. Palmer (1991). Introduction to the Theory of Neural Computation, Reading : Addison-Wesley.

[7] Jones, A.J. (1993). The Modular Construction of Dynamic Nets, Neural Computing and Applications, 1, pp. 91-95, London : Springer-Verlag.

[8] Jones, A.J. (1993). Genetic Algorithms and Their Applications to the Design of Neural Networks, Neural Computing and Applications, 1, pp. 32-45, London : Springer-Verlag.

[9] Kohonen, T. (1989). Self-Organization and Associative Memory, Berlin : Springer-Verlag, 3rd Ed.

[10] Kohonen, T. (1990). The Self-Organizing Map. Proceedings of the IEEE, 78(9), pp. 1464-1480.

[11] Kohonen, T. (1991). Self-Organizing Maps : Optimization Approaches. In Proc. of the Int. Conf. on ANNs, pp. 981-990, Espoo, Finland.

[12] Kohonen, T., J. Kangas, and J. Laaksonen (1992). User Manual of SOM-PAK - The Self-Organizing Map Program Package, Helsinki University of Tech., Laboratory of Comp. and Info. Sci., Espoo, Finland.

[13] McClelland, J.L., and A.K. Kawamoto (1986). Mechanics of Sentence Processing: Assigning Roles to Constituents of Sentences. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition, V.2: Psychological and Biological Models, eds. J.L. McClelland, D.E. Rumelhart, and the PDP Research Group, Cambridge : MIT Press.

[14] Waltz, D.L., and J.B. Pollack (1988). Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation. In Connectionist Models and Their Implications: Readings from Cognitive Science, eds. D.L. Waltz, and J.A. Feldman, Norwood : Ablex Pub. Corp.

Appendix A : Sample Problems

1. Sample Problem 1 : (Taxonomy of Abstract Data) The hierarchical structure of data in a tree structure and the input data matrix are given below. The input data matrix contains 32 patterns with 5 attributes each.

Tree Structure

```
A-B-C-D-E
    |
    F
    |
    G
    |
    H-K-L-M-N-O-P-Q-R
    |     |     |
    I     S     W
    |     |     |
    J     T     X-1-2-3-4-5-6
          |     |
          U     Y
          |     |
          V     Z
```

Input Data Matrix

Items	Attributes					Items	Attributes				
A	1	0	0	0	0	Q	3	3	7	0	0
B	2	0	0	0	0	R	3	3	8	0	0
C	3	0	0	0	0	S	3	3	3	1	0
D	4	0	0	0	0	T	3	3	3	2	0
E	5	0	0	0	0	U	3	3	3	3	0
F	3	1	0	0	0	V	3	3	3	4	0
G	3	2	0	0	0	W	3	3	6	1	0
H	3	3	0	0	0	X	3	3	6	2	0
I	3	4	0	0	0	Y	3	3	6	3	0
J	3	5	0	0	0	Z	3	3	6	4	0
K	3	3	1	0	0	1	3	3	6	2	1
L	3	3	2	0	0	2	3	3	6	2	2
M	3	3	3	0	0	3	3	3	6	2	3
N	3	3	4	0	0	4	3	3	6	2	4
O	3	3	5	0	0	5	3	3	6	2	5
P	3	3	6	0	0	6	3	3	6	2	6

2. Sample Problem 2 : The features for the nouns are given below. The numbers indicate the numerical value of the feature. There are 32 patterns with 9 attributes each.

Feature Dimension	Values for Nouns
HUMAN	human=1 nonhuman=2
SOFTNESS	soft=1 hard=2
GENDER	male=1 female=2 neuter=3
VOLUME	small=1 medium=2 large=3
FORM	compact=1 1-D=2 2-D=3 3-D=4
POINTNESS	pointed=1 rounded=2
BREAKABILITY	fragile=1 unbreakable=2
OBJECT TYPE	food=1 toy=2 tool=3 utensil=4 furniture=5 animate=6 natural-inanimate=7

0 means "non-applicable". The format of the attributes is [HUMAN, SOFTNESS, GENDER, VOLUME, FORM, POINTNESS, BREAKABILITY, OBJECT TYPE]. Explanation for some nouns : ft_bat : flying bat, bb_bat : baseball bat, paperwt : paperweight, li_chicken : living chicken, co_chicken : cooked chicken.

Input Data Matrix

Nouns	Attributes	Nouns	Attributes
ball	2 1 3 1 1 2 2 2	hatchet	2 2 3 1 2 1 2 3
fl_bat	2 1 1 1 4 1 2 6	hammer	2 2 3 1 2 2 2 3
bb_bat	2 2 3 1 2 2 2 2	man	1 1 1 3 4 2 2 6
bat	2 0 0 1 0 0 2 0	woman	1 1 2 3 4 2 2 6
boy	1 1 1 2 4 2 2 6	plate	2 2 3 1 3 2 1 5
paperwt	2 2 3 1 1 1 2 5	rock	2 2 3 1 4 1 2 7
cheese	2 1 3 1 3 2 1 1	potato	2 1 3 1 1 2 1 1
li_chicken	2 1 2 1 4 2 1 6	pasta	2 1 3 1 2 2 1 1
co_chicken	2 1 3 1 1 2 1 1	spoon	2 2 3 1 2 2 2 4
chicken	2 1 0 1 0 2 0 0	carrot	2 2 3 1 2 1 1 1
curtain	2 1 3 2 3 2 1 5	vase	2 2 3 1 2 2 1 5
desk	2 2 3 3 4 1 2 5	window	2 2 3 2 3 1 1 5
doll	2 1 2 1 4 2 1 2	dog	2 1 1 2 4 2 2 6
food	2 1 3 1 0 2 1 1	wolf	2 1 1 2 4 1 2 6
fork	2 2 3 1 2 1 2 4	sheep	2 1 2 2 4 2 1 6
girl	1 1 2 2 4 2 2 6	lion	2 1 1 3 4 1 2 6

3. Sample Problem 3 : The concepts and the features (or attributes) are given below. There are 10 patterns with 29 attributes each. Concepts : weekend=A, outdoors=B, casino=C, video games=D, fire at=E, waste money=F, deer=G, dollar=H, hunting=I, and gambling=J. The letters associated with the concepts are used for abbreviation. The meaning of the numbers in the following table is : 1=positive association, 0.5=mild association, -0.5=negative association, and 0=unrelated.

Input Data Matrix

Features	Concepts A	B	C	D	E	F	G	H	I	J
second	0.0	0.0	0.0	0.5	1.0	0.0	0.0	0.0	0.0	0.0
minute	0.0	0.0	0.5	1.0	0.0	0.0	0.0	0.0	0.0	0.5
hour	0.0	0.0	1.0	0.5	0.0	0.0	0.0	0.0	0.5	1.0
day	1.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	1.0	0.5
week	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
month	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
year	-0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
decade	-0.5	0.0	-0.5	-0.5	0.0	0.0	0.0	0.0	-0.5	-0.5
inside	0.0	0.0	1.0	1.0	0.0	0.5	0.0	0.0	0.0	1.0
house	1.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.5
store	1.0	-0.5	0.5	0.5	0.5	1.0	-0.5	1.0	0.0	0.0
office	0.0	-0.5	0.0	-0.5	-0.5	0.0	-0.5	1.0	-0.5	-0.5
school	-0.5	0.0	-0.5	-0.5	-0.5	-0.5	0.0	0.0	-0.5	-0.5
factory	0.0	-0.5	-0.5	-0.5	-0.5	0.0	-0.5	1.0	-0.5	-0.5
casino	0.5	-0.5	1.0	0.5	0.0	1.0	-0.5	1.0	-0.5	1.0
bar	1.0	-0.5	0.5	0.5	0.0	1.0	-0.5	0.5	0.0	0.5
restaurant	0.5	-0.5	0.5	0.0	-0.5	0.5	-0.5	0.5	0.0	0.5
theater	1.0	-0.5	0.5	0.0	-0.5	0.5	-0.5	0.5	-0.5	-0.5
outside	1.0	1.0	-0.5	-0.5	0.5	0.0	1.0	0.0	1.0	0.5
racetrack	0.5	0.5	0.5	0.0	0.0	1.0	-0.5	1.0	-0.5	1.0
city street	0.0	0.5	0.5	0.0	0.5	0.0	-0.5	0.5	-0.5	0.0
city park	0.5	0.5	-0.5	-0.5	0.5	0.0	0.5	0.5	0.0	0.0

rural	0.5	0.5	-0.5	-0.5	0.5	0.0	1.0	0.5	0.5	0.0
forest	0.5	0.5	-0.5	-0.5	0.5	-0.5	1.0	-0.5	1.0	0.0
lake	0.5	0.5	0.0	-0.5	0.5	0.0	0.5	0.0	0.5	0.0
desert	0.5	0.5	0.5	-0.5	0.5	-0.5	0.0	-0.5	0.5	0.0
mountain	0.5	0.5	0.0	-0.5	0.5	-0.5	0.0	-0.5	0.5	0.0
seashore	0.5	0.5	0.0	-0.5	0.0	0.0	-0.5	0.0	-0.5	0.0
canyon	0.5	0.5	-0.5	-0.5	0.0	0.0	0.0	0.0	0.0	0.0

4. Sample Problem 4 : The nouns and their attributes are given below. There are 27 patterns with 7 attributes each. The ordinal value of a letter is according to English Alphabet.

Input Data Matrix

Nouns	Ordinal value of letters							Nouns	Ordinal value of letters						
ball	2	1	12	12	0	0	0	woman	23	15	13	1	14	0	0
bat	2	1	20	0	0	0	0	plate	16	12	1	20	5	0	0
boy	2	15	25	0	0	0	0	rock	18	15	3	11	0	0	0
cheese	3	8	5	5	19	5	0	potato	16	15	20	1	20	15	0
chicken	3	8	9	3	11	5	14	pasta	16	1	19	20	1	0	0
curtain	3	21	18	20	1	9	14	spoon	19	16	15	15	14	0	0
desk	4	5	19	11	0	0	0	carrot	3	1	18	18	15	20	0
doll	4	15	12	12	0	0	0	vase	22	1	19	5	0	0	0
food	6	15	15	4	0	0	0	window	23	9	14	4	15	23	0
fork	6	15	18	11	0	0	0	dog	4	15	7	0	0	0	0
girl	7	9	18	12	0	0	0	wolf	23	15	12	6	0	0	0
hatchet	8	1	20	3	8	5	18	sheep	19	8	5	5	16	0	0
hammer	8	1	13	13	5	18	0	lion	12	9	15	14	0	0	0
man	13	1	14	0	0	0	0								

5. Sample Problem 5 : There are 400 points generated at random in a unit square. The points are uniformly distributed.

6. Sample Problem 6 : There are 400 points generated at random in a plus-shaped polygon in a unit square. The points are uniformly distributed.

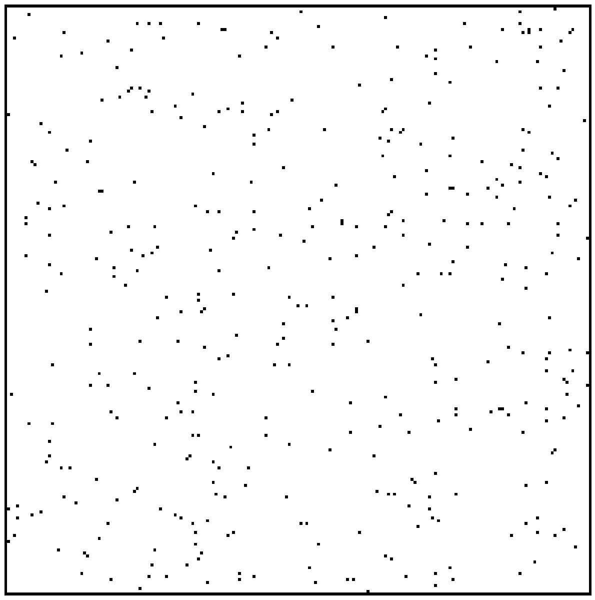

Figure 10: The Fifth Sample Problem.

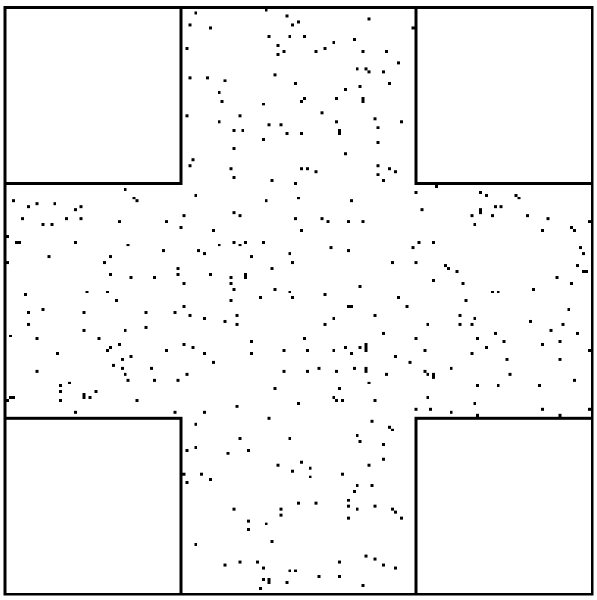

Figure 11: The Sixth Sample Problem.