Two Efficient Connectionist Schemes for Structure Preserving Dimensionality Reduction

Nikhil R. Pal, *Member, IEEE,* and Vijay Kumar Eluri

***Abstract*—We propose two neural-net-based methods for structure preserving dimensionality reduction. Method 1 selects a small representative sample and applies Sammon's method to project it. This projected data set is then used to train an MLP. Method 2 uses Kohonen's self-organizing feature map (SOFM) to generate a small set of prototypes which is then projected by Sammon's method. This projected data set is then used to train an MLP. Both schemes are quite effective in terms of computation time and quality of output, and both outperform methods of Jain and Mao on the data sets tried.**

***Index Terms*—Connectionist models, dimensionality reduction, feature extraction, multilayer perceptron, self-organizing feature map.**

I. Introduction

FEATURE extraction and dimensionality reduction are two important problems in pattern recognition and exploratory data analysis. Feature extraction can avoid the "curse of dimensionality," improve generalization ability of classifiers by eliminating harmful features, and reduce the space and computational requirements associated with analyzes of the data. Many features not only lead to more computational overhead but often it can create confusion thereby degrading the performance of a classifier designed on them.

Dimensionality reduction can be done mainly in two ways: selecting a small but important subset of features and generating (extracting) a lower dimensional data preserving the distinguishing characteristics of the original higher dimensional data. Dimensionality reduction not only helps in the design of a classifier, it also helps in other exploratory data analysis. It can help in both clustering tendency assessment as well as to decide on the number of clusters by looking at the scatter plot of the lower dimensional data.

Object data are represented as $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, a set of n feature vectors (signals) in $\mathcal{R}^p$. The jth observed object (some physical entity such as a tank, image, patient, stock market report, etc.) has vector $\mathbf{x}_j$ as its numerical representation; x_{jk} is the kth characteristic (or feature) associated with object j. Feature extraction and data projection can be viewed as an implicit or explicit mapping Φ from a p-dimensional input space to a q-dimensional output space

$$\Phi: \mathcal{R}^p \rightarrow \mathcal{R}^q \tag{1}$$

such that some criterion, $\mathcal{C}$, is optimized. Usually $q \leq p$, but for some feature extraction problems q may be greater than p also. This formulation is similar to a function approximation problem. However, unlike the function approximation problem, where the mapping function is estimated from training patterns which are input–output pairs (desired outputs are known), in feature extraction and data projection the desired outputs are not available. In effect, any feature extraction method has to produce $Y = \Phi(X) \subset \mathcal{R}^q$.

A large number of approaches for feature extraction and data projection are available in the pattern recognition literature [1], [2]. These approaches differ from each other in the characteristics of the mapping function Φ, how Φ is learned, and what optimization criterion $\mathcal{C}$ is used. The mapping function can be either *linear* or *nonlinear*, and can be learned through either *supervised* or *unsupervised* methods.

In this paper we propose schemes for structure preserving dimensionality reduction. The first method integrates the theory of statistical subsampling, structure preserving characteristic of Sammon's function and the generalization capability of multilayer neural networks; while the second scheme combines Sammon's function, self-organizing feature maps along with multilayer perceptron networks. Both methods can produce, like Sammon's algorithm, lower dimensional data which are coherent with the original data at a much lower computational cost. Both methods have been compared with original Sammon's algorithm and a connectionist method due to Jain and Mao.

The remaining part of this paper is organized as follows: Section II deals with conventional methods while Section III considers connectionist methods. In Section IV, we present our proposed methods. In Section V, we discuss the implementation and the results obtained. The paper is concluded in Section VI.

II. Conventional Methods for Data Projection

We describe here two unsupervised methods for data projection: Principal Component Analysis (PCA) and Sammon's nonlinear projection method.

A. Principal Component Analysis

Principal component analysis (PCA) [1] is a well-known widely used statistical technique for feature extraction and data compression. It appears under various names, including the Karhunen–Loeve transform in signal processing and the Hotelling transform in image processing. It is a linear or-

Manuscript received November 20 1997; revised June 30, 1998.

N. R. Pal is with the Machine Intelligence Unit, Indian Statistical Institute, Calcutta, 35, India.

V. K. Eluri is with the Vedika International Pvt Ltd., Calcutta, 20, India.

Publisher Item Identifier S 1045-9227(98)07349-4.

thogonal transform from p-dimensional space to q-dimensional space ($q \leq p$), such that the coordinates of the data in the new q-dimensional space are uncorrelated and maximal amount of variance of the original data is preserved by only a small number of coordinates.

Suppose we have a linear transform from a p-dimensional zero-mean input vector $\mathbf{x} = (x_1, x_2, \cdots, x_p)^T$ to a q-dimensional output vector $\mathbf{y}$, where $\mathbf{y} = (y_1, y_2, \cdots, y_q)^T$ and $\mathbf{y}$ is related to $\mathbf{x}$ by the expression $\mathbf{y} = W\mathbf{x}$, W is a $q \times p$ matrix, with $q \leq p$. PCA sets the q successive rows of W to the q eigenvectors corresponding to the q largest eigenvalues of the input covariance matrix $S = E(\mathbf{x}\mathbf{x}^T)$. Thus y_1 represents the component of $\mathbf{x}$ in the direction of the largest eigenvector of S, y_2 is the component in the direction of the second largest, and so on. The computation of principal components is summarized in the following algorithm:

PCA( )
{
Input p, q and $X = \{\mathbf{x}_i \in \mathcal{R}^p: i = 1, 2, \cdots, n\}$;
$\mu = \frac{1}{n}\sum_{k=1}^{n} \mathbf{x}_k$;
$S = \frac{1}{n}\sum_{k=1}^{n} (\mathbf{x}_k - \mu)(\mathbf{x}_k - \mu)^T$;
Compute and order the eigenvalues
$\lambda_1 \geq \lambda_2 \geq \cdots \lambda_p > 0$ of S ;
Construct the associated (ordered) orthogonal
eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p$;
Form the matrix $W^T = [\mathbf{v}_1 \mid \mathbf{v}_2 \mid \cdots \mid \mathbf{v}_q]$;
Compute $\mathbf{y}_k = W\mathbf{x}_k \, for \, k = 1, 2, \cdots, n$;
}.

Note that W can be used with any new vector $\mathbf{x}_{n+1}$ that has not been used to compute the projection. Let W_n and W_{n+1} be the W matrices computed with $X_n = \{\mathbf{x}_i \in \mathcal{R}^p: i = 1, 2, \cdots, n\}$ and $X_{n+1} = X_n \cup \{\mathbf{x}_{n+1} \in \mathcal{R}^p\}$, respectively. Then

$$\mathbf{y}_{n+1} = W_{n+1}\mathbf{x}_{n+1} \neq W_n\mathbf{x}_{n+1} = \mathbf{y'}_{n+1}.$$

As long as $\mathbf{x}_{n+1}$ is not widely different from the vectors used to compute W_n, $\mathbf{y}_{n+1} \approx \mathbf{y'}_{n+1}$. Thus W_n can be used to project new data points, as long as the data points used to compute W_n adequately represent the population generating $\mathbf{x}_i \in \mathcal{R}^p$.

B. Sammon's Nonlinear Projection Method

Sammon [3] proposed a simple yet very useful nonlinear projection algorithm that attempts to preserve the structure by finding n points in q-dimensional space such that interpoint distances approximate the corresponding interpoint distances in p-dimensional space.

Let $X = \{\mathbf{x}_k | \mathbf{x}_k = (x_{k1}, x_{k2}, \cdots, x_{kp})^T, k = 1, 2, \cdots, n\}$ be the set of n input vectors and let $Y = \{\mathbf{y}_k | \mathbf{y}_k = (y_{k1}, y_{k2}, \cdots, y_{kq})^T, k = 1, 2, \cdots, n\}$ be the unknown vectors to be found.

Let $d^*_{ij} = d(\mathbf{x}_i, \mathbf{x}_j), \mathbf{x}_i, \mathbf{x}_j \in X$ and $d_{ij} = d(\mathbf{y}_i, \mathbf{y}_j), \mathbf{y}_i, \mathbf{y}_j \in Y$, where $d(\mathbf{x}_i, \mathbf{x}_j)$ be the Euclidian distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. Sammon suggested looking for Y minimizing the error function E

$$E = \frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}. \tag{2}$$

Minimization of E is an unconstrained optimization problem in the nq variables y_{ij}, $i = 1, 2, \cdots, n$; $j = 1, 2, \cdots, q$. Sammon used the method of steepest descent for (approximate) minimization of E. Let $\mathbf{y}_i(t)$ to be the estimate of $\mathbf{y}_i$ at the tth iteration, $\forall i$. Then $\mathbf{y}_i(t+1)$ is given by

$$y_{ij}(t+1) = y_{ij}(t) - \alpha \left[\frac{\frac{\partial E(t)}{\partial y_{ij}(t)}}{\left| \frac{\partial^2 E(t)}{\partial y_{ij}(t)^2} \right|} \right] \tag{3}$$

where the nonnegative scalar constant α (Sammon called it a magic factor and recommended $\alpha \approx 0.3$ or 0.4) is the step size for gradient search. Now

$$\frac{\partial E(t)}{\partial y_{ij}(t)} = -\frac{2}{\lambda} \sum_{k=1, k\neq i}^{n} \left[\frac{(d^*_{ik} - d_{ik})}{(d^*_{ik} \cdot d_{ik})} \right] (y_{ij} - y_{kj}) \tag{4}$$

and

$$\frac{\partial^2 E(t)}{\partial y_{ij}(t)^2} = \frac{-2}{\lambda} \sum_{k=1, k\neq i}^{n} \left[\frac{1}{d^*_{ik} d_{ik}} \right] \left[(d^*_{ik} - d_{ik}) - \left(\frac{(y_{ij} - y_{kj})^2}{d_{ik}} \right) \left(1 + \frac{(d^*_{ik} - d_{ik})}{d_{ik}} \right) \right] \tag{5}$$

where $\lambda = \sum_{i<j} d^*_{ij}$.

Most authors follow Sammon in continuing to exhibit λ in deference to the original formulation. However, it is not necessary to maintain λ in (4) and (5) for a successful solution of the optimization problem, since minimization of $(1/\sum_{i<j} d^*_{ij}) \sum_{i<j} ((d^*_{ij} - d_{ij})^2 / d^*_{ij})$ and $\sum_{i<j} ((d^*_{ij} - d_{ij})^2 / d^*_{ij})$ yield the same solutions.

Our implementation of Sammon's algorithm is shown at the bottom of the next page.

With this method we cannot get an explicit mapping function governing the relationship between patterns in p-space and corresponding patterns in q-space. Therefore, it is not possible to project new points. This method also involves a large amount of computation, as every step within an iteration requires the computation of $n(n-1)/2$ distances. The algorithm becomes impractical for large n. Finally, there are many local minima on the error surface and it is usually unavoidable for the algorithm to get stuck in some local minimum.

There have been a few attempts [4]–[7] to reduce the computational overhead of Sammon's method using concepts of clustering. For example, Schachter [4] partitioned the feature space into a number of bins (hyper-rectangles). Let $\mathbf{z}_i$ be the location of the ith bin and w_i be the number of points in the ith bin. (In [4] it was not clearly spelled out how $\mathbf{z}_i$ was computed, we assume that it was the centroid of the associated bin.) Instead of using the entire data set, Schachter projected

only the $\mathbf{z}_i$s where the projected vectors are learned using an update rule which is almost like Sammon's update rule. We say "almost like" because the learning coefficient for updating the pair of vectors corresponding to bins $\mathbf{z}_i$ and $\mathbf{z}_j$ were taken to be proportional to $w_i/(w_i+w_j)$ and $w_j/(w_i+w_j)$, and in the denominator of the update expression there was an extra term d_{ij}. This is a heuristic update rule and may not minimize Sammon's error function.

III. Connectionist Methods for Data Projection

The potential of artificial neural networks (ANN's) in various applications is well established. Such massively parallel computing models are being widely used in many branches of applied computer science. One interesting feature of such an ANN is its ability to learn from the environment in an adaptive manner. Recently a large number of ANN's and associated learning algorithms have been proposed for feature extraction and multivariate data projection [8]–[20]. These networks exhibit some nice properties which are different from classical approaches. Although these methods do not necessarily provide new approaches to feature extraction and data projection (from the viewpoint of functionality performed by the networks), they do possess some advantages over traditional approaches: 1) Most learning algorithms and neural networks are adaptive in nature, thus they are well-suited for many real environments where adaptive systems are required. 2) For real-time implementation, neural networks provide good, if not the best, architectures which can be easily implemented using current very lage scale integration (VLSI) and optical technologies. 3) Neural-network implementations offer generalization ability for projecting new data.

A. PCA Networks

Due to unavoidable computational complexity with the conventional matrix algebraic approaches, especially when p is very large, the neural-network approach for PCA has been widely studied recently. A variety of neural networks and learning algorithms have been proposed for PCA [12]–[14]. Most of them are based on the early work of Oja's one-unit algorithm [10], [11]. We discuss here only one of them that has been used in our study.

Sammon_Projection( )
{
  Input p, q and $X = \{\mathbf{x}_i \in \mathcal{R}^p: i = 1, 2, \cdots, n\}$;
  Input $\epsilon > 0$; /* Limit on Sammon's error */
  Input maxstep; /* Maximum number of updating steps */
  Generate randomly $Y = \{\mathbf{y}_i \in \mathcal{R}^q: i = 1, 2, \cdots, n\}$;
  Compute $D^* = [d^*_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)]_{n\times n}$;
  Compute $D = [d_{ij} = d(\mathbf{y}_i, \mathbf{y}_j)]_{n\times n}$;
  error = High value; /* Any arbitrary large value */
  $t = 1$;
  while((error $> \epsilon$) and ($t \le$ maxstep))
  {
    for($i = 1$; $i \le n$; $i++$)
    {
      for($j = 1$; $j \le q$; $j++$)
      {
        Compute $\dfrac{\partial E(t)}{\partial y_{ij}(t)}$ using (4);
        Compute $\dfrac{\partial^2 E(t)}{\partial y_{ij}(t)^2}$ using (5);
        $$y_{ij}(t+1) = y_{ij}(t) - \alpha \left[\frac{\dfrac{\partial E(t)}{\partial y_{ij}(t)}}{\left| \dfrac{\partial^2 E(t)}{\partial y_{ij}(t)^2} \right|} \right];$$
      }
    }
    Compute $D = [d_{ij} = d(\mathbf{y}_i, \mathbf{y}_j)]_{nXn}$;
    $$error = \frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}};$$
    $t = t + 1$;
  }
}

B. Rubner's PCA Network

The PCA network proposed by Rubner and Schulten [12] and Rubner and Tavan [13] consists of an input layer with p nodes and an output layer with q nodes. The two layers are completely interconnected. The connection weight between input node i and output node j is denoted by w_{ji}. All output nodes are hierarchically organized in such a way that the output node i is connected to the output node j with connection strength u_{ij} if and only if $j < i$.

The set of weights connecting an output node j to all input nodes forms the weight vector $\mathbf{w}_j$, the transpose of which is the jth row of the weight matrix W. Given the input vector $\mathbf{x}_k \in X$, the corresponding output vector $\mathbf{y}_k \in Y$ is computed by the network as

$$\mathbf{y}_k = \left\{ y_{kj} = \langle \mathbf{w}_j, \mathbf{x}_k \rangle + \sum_{l<j} (u_{jl} \times y_{kl}),\ j = 1, \cdots, q \right\}^T. \tag{6}$$

The weights between the two layers are adjusted upon presentation of an input pattern $\mathbf{x}_k$ according to the Hebbian rule,

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta \mathbf{x}_k y_{kj} \qquad (j = 1, \cdots, q). \tag{7}$$

The lateral weights adapt themselves according to the anti-Hebbian rule,

$$u_{jl}(t+1) = u_{jl}(t) - \mu y_{kj} y_{kl} \qquad (l < j) \tag{8}$$

where η and μ are positive learning coefficients. Note that (7) updates a complete weight vector, while (8) updates only one weight at a time.

Rubner and Tavan [13] proved that if the learning parameters η and μ are chosen according to

$$\frac{\eta(\lambda_1 - \lambda_p)}{\lambda_1(1 + \eta\lambda_p)} < \mu < \frac{2}{\lambda_1} \tag{9}$$

then the learning rules (7) and (8) force the lateral weights to vanish and the activities of the output cells to become uncorrelated. Consequently, the weight vectors $\mathbf{w}_j$ converge to the eigenvectors of the covariance matrix S. Although in practice, it is difficult to determine the values of η and μ according to the inequality (9) without computing the eigenvalues, (9) does provide a range for the values of η and μ if λ_1 and λ_p can somehow be estimated.

As in [9] the convergence rate can be improved introducing a momentum term and letting the learning rate and momentum decay with time. Equations (7) and (8) with momentum term can be rewritten as

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta(t)\mathbf{x}_k y_{kj} + \beta(t)\triangle\mathbf{w}_j(t) \tag{10}$$

and

$$u_{jl}(t+1) = u_{jl}(t) - \mu(t) y_{kj} y_{kl} + \beta(t)\triangle u_{jl}(t) \tag{11}$$

where

$$\eta(t+1) = \max(\alpha * \eta(t), 0.001) \tag{12}$$

$$\mu(t+1) = \max(\alpha * \mu(t), 0.002) \tag{13}$$

$$\beta(t+1) = \max(\alpha * \beta(t), 0.001). \tag{14}$$

In (10)–(14) t is the iteration index and α is the decay rate.

We used the following implementation of the PCA network as shown at the bottom of the next page.

The PCA network has the same level of generalization abilities as that of W computed in Section II-A and hence is able to project new data as expected when the original data have linear relationship. *However, PCA networks and learning algorithms have some limitations that diminish their attractiveness: 1) PCA networks are able to realize only linear input-output mappings and 2) PCA networks cannot usually separate independent subsignals from their linear mixture.*

To overcome these drawbacks PCA networks containing nonlinear units are gaining attention [21], [22]. Also independent component analysis (ICA) has been introduced as an interesting extension of PCA in the context of signal separation problem [23].

C. Neural-Network Architecture for Sammon's Projection

Let us express Sammon error given in (2) as

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E_{ij}$$

where

$$E_{ij} = \lambda \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}$$

and

$$\lambda = \frac{1}{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d^*_{ij}}.$$

d^*_{ij} and d_{ij} are the distances, respectively, in the input space and in the projected space between patterns i and j.

Jain and Mao [8], [9] proposed an interesting multilayer feedforward network for Sammon's projection. The number of input nodes is set to p. The number of output nodes is equal to q. Let $X = \{\mathbf{x}_k | \mathbf{x}_k = (x_{k1}, x_{k2}, \cdots, x_{kp})^T, k = 1, 2, \cdots, n\}$ be the set of n p-dimensional input vectors. The output of the tth node in layer ℓ is denoted by $y^{(\ell)}_{it}$, $t = 1, 2, \cdots, n_\ell$; $\ell = 0, 1, \cdots, L$, where n_ℓ is the number of nodes in layer ℓ, L is the number of layers, and $y^{(0)}_{it} = x_{it}$, $t = 1, 2, \cdots, p$. The connection weight between node s in layer $\ell - 1$ and node t in layer ℓ is represented by $w^{(\ell)}_{ts}$. The output of the tth node in layer ℓ is given by

$$y^{(\ell)}_{it} = f\left(\sum_{s=1}^{n_{\ell-1}} w^{(\ell)}_{ts} y^{(\ell-1)}_{is} \right), \qquad \ell = 1, 2, \cdots, L. \tag{15}$$

Using the above notation, d_{ij} can be written as

$$d_{ij} = \left\{ \sum_{u=1}^{q} [y_{iu}^{(L)} - y_{ju}^{(L)}]^2 \right\}^{1/2}.$$

Using backpropagation of errors, the update rule for the output layer is given by

$$\begin{aligned} \triangle w_{ut}^{(L)} &= -\eta \frac{\partial E_{ij}}{\partial w_{ut}^{(L)}} \\ &= -\eta \Big(\triangle_{iut}^{(L)} y_{it}^{(L-1)} - \triangle_{jut}^{(L)} y_{jt}^{(L-1)} \Big) \end{aligned} \tag{16}$$

where η is the learning rate.

Similarly, the update rule for all hidden layers, $\ell = 1, 2, \cdots, L-1$ is given by

$$\begin{aligned} \triangle w_{ts}^{(\ell)} &= -\eta \frac{\partial E_{ij}}{\partial w_{ts}^{(\ell)}} \\ &= -\eta \Big(\triangle_{its}^{(\ell)} y_{is}^{(\ell-1)} - \triangle_{jts}^{(\ell)} y_{js}^{(\ell-1)} \Big) \end{aligned} \tag{17}$$

where

$$\begin{aligned} \triangle_{its}^{(\ell)} &= \delta_{it}^{(\ell)} [1 - y_{is}^{(\ell)}] y_{is}^{(\ell)} \\ \triangle_{jts}^{(\ell)} &= \delta_{jt}^{(\ell)} [1 - y_{jt}^{(\ell)}] y_{jt}^{(\ell)} \\ \delta_{it}^{(\ell)} &= \sum_{u=1}^{m} \triangle_{iut}^{(\ell+1)} w_{ut}^{(\ell+1)} \end{aligned}$$

and

$$\delta_{jt}^{(\ell)} = \sum_{u=1}^{m} \triangle_{jut}^{(\ell+1)} w_{ut}^{(\ell+1)}.$$

Next we provide a schematic description of the Sammon_Network [8] as shown at the bottom of the page.

In [8], it was experimentally shown that two layered networks perform better than three layered networks. It was also shown experimentally that results can be improved by increasing the number of free parameters. *The system requires a lot of space and training time to get good results. The error after training, as we will see, is also not found comparable to that of Sammon's algorithm.*

In [9] another approach was followed for training so as to take advantage of the nonlinearity of the above network. Initially the PCA network is used to project data and then standard backpropagation algorithm is used to approximate principal components. The connection weights of such a trained MLP are then used to initialize the weights of the Sammon's neural net. This means that Sammon's network is initialized such that it behaves like a PCA network.

This implementation style has a number of disadvantages: 1) training time is long; 2) memory usage is high; 3) to try a different (new) architecture, again an MLP with the same (new) architecture should be trained to approximate principal components for initialization of the weights of Sammon_Network, i.e., we cannot add any extra hidden layer even if it is demanded, in fact we cannot even add an extra node; and finally 4) the main purpose of this network is to handle nonlinear data,[1] *as, linear data is very well projected by the PCA network, but, even this may not be achieved by the proposed three-step (PCA-MLP-*

[1]Loosely we call a data set linear, if a linear transformation can project it to a lower dimension preserving the cluster structure in the original data; otherwise, the data set is nonlinear in nature.

PCA_Network( )
{
  Input p, q and $X = \{\mathbf{x}_i \in \mathcal{R}^p : i = 1, 2, \cdots, n\}$;
  Input $\epsilon > 0$; /* Threshold for lateral weights */
  Input maxstep; /* Maximum number of iterations allowed */
  Input η, μ; /* Learning Rates for w_{ij} and u_{jl} */
  Input β, α; /* Momentum and decay rate */
  Generate initial weights randomly; /* w_{ji} and u_{jl} */
  $t = 1$;
  flag = true;
  while(($t \leq$ maxstep) and (flag = true))
  {
    Randomly select an input pattern $\mathbf{x}_k$;
    Compute output vector $\mathbf{y}_k$ using (6);
    Update connection weights between the input nodes
    and output nodes according to (10);
    Update lateral weights according to (11);
    Normalize each weight vector $\mathbf{w}_j$ to unit norm;
    Modify η, μ and β according to (12), (13), (14) respectively;
    $t = t + 1$;
    if ($|u_{jl}| < \epsilon \forall_{jl}$ $(j < l)$)
      flag = false;
  }
}

Sammon_Net) implementation [9], as shall be seen from the results section. Note that all results reported in Section V are obtained by this three-step implementation.

IV. Proposed Methods

A. Method 1

Sammon's algorithm and some of its derivatives work very well for small data sets. But, these methods involve a large amount of computation, as every step within an iteration requires the computation of $n(n-1)/2$ distances. Therefore, these algorithms quickly become impractical for large n. Moreover, as mentioned earlier, Sammon's algorithm cannot project new data points; with every new data point the entire augmented data set is to be reprojected. These problems can be eliminated, if we can get a mapping function governing the relationship between patterns in p-dimensional space and patterns in q-dimensional space, by projecting a small representative subset of the data. The various implementations of Sammon's algorithm [4], [7] do not give an explicit mapping function representing the relationship between patterns in p-dimensional space and patterns in q-dimensional space.

We propose here a very simple method, which performs better (at least on the examples we tried) than methods given in [8], [9] in terms of *time*, *space*, and *quality of the projected map*. This method combines the statistical theories of subsampling along with the advantages of Sammon's method for projecting small data sets and capabilities of MLP for function approximation, as it is known that backpropagation networks can approximate any square integrable function to any desirable degree of accuracy [24]–[27].

When we talk about projection of unknown data based on a mapping (explicit or implicit) estimated from a given data set we implicitly assume that the given data have some structure which future data points are expected to follow. This is natural, otherwise, prediction does not have any meaning. In other words, we can assume that the data points are generated from some (unknown) time-invariant probability distribution. Therefore, if we can extract a small but an adequate representative sample of the given data set and then estimate the mapping function based on these we can expect to have a good generalization.

In fact although not explained or stated explicitly this was the philosophy behind Jain's method also. It then raises two issues: How to get an adequate but small sample and what do we do with that. For the time being let us assume that we are provided with a solution to the first issue (we discuss it later), i.e., we are given a small representative data set $X^{(S)}$. Now we propose to run Sammon's algorithm on $X^{(S)}$ to generate $Y^{(S)} \subset \mathcal{R}^q$. Then we use $(X^{(S)}, Y^{(S)})$ to train an MLP. Note that such a trained MLP will capture the structure present in $(X^{(S)}, Y^{(S)})$. In [9] PCA net was used to initialize Sammon's net but PCA is a linear mapping which has, as such, no relation to Sammon's projection. Finally, in [8] and [9], Sammon's network is used to extract the relation which, as their results reveal, demands too many free parameters to get a reasonably good solution. While in our scheme the relation is already captured by the pair $(X^{(S)}, Y^{(S)})$ and MLP simply learns it. Hence, as will be seen later, our scheme is expected to get much better results at a lower cost.

Sammon_Network( )
{
  Input p, q and $X = \{\mathbf{x}_l \in \mathcal{R}^p : l = 1, 2, \cdots, n\}$;
  Input $\epsilon > 0$; /* Limit of Sammon's error */
  Input maxstep; /* Maximum number of updating steps; */
  Input η; /* Learning rate */
  error = High value; /* Any arbitrary large value */
  $t = 1$;
  Generate the initial weights randomly
  while((error $> \epsilon$) and ($t \leq$ maxstep))
  {
    Repeat k times
      Select a pair of patterns i and j randomly;
      Present i and j to the network one at a time;
      Compute outputs of all nodes using (15);
      Update connection weights according to (16) and (17);
    End Repeat
    Present all patterns, compute outputs of the network

    Calculate $error = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E_{ij}$;

    $t = t + 1$;
  }
}

Shankar and Pal [28] demonstrated that it is possible to obtain a small but adequate representative sample that captures the basic characteristics of the original data. For the present case we can have a low cost scheme for selection of a small subset $X^{(S)}$ of representative data points based on χ^2 or divergence statistic so that statistical characteristics of X are retained by $X^{(S)}$. The scheme follows like this: define a frequency distribution (f) on X. Draw a small random sample $X^{(S)}$ with p% of the data points in X. We use the simple random sampling without replacement (SRSWOR) scheme. Compute the frequency distribution f_S on $X^{(S)}$. Now to check how good $X^{(S)}$ captures the structure present in X, we find the agreement between f_S and f by χ^2 or divergence statistic. If the agreement is not acceptable at some predefined level α, we append $X^{(S)}$ by an additional random sample of l% to get the new $X^{(S)}$ = old $X^{(S)}$+ the extra l% sample. The hypothesis of satisfactory agreement between f_S and f is again tested. The process is repeated till the hypothesis is accepted. It has been empirically observed that 30% data points are usually enough for the purpose of preserving cluster substructures. In the present case, this result is applicable too. However, to keep consistency with the method of Jain and Mao, we used 50% data (note that in [9] 500 data points were taken for data sets of size 1000) for all computational experiments reported here.

Next we provide a schematic description of the proposed method:

Method-1( )
{
Input $X = \{\mathbf{x}_i \in \mathcal{R}^p: i = 1, 2, \cdots, n\}$;
Normalize $\mathbf{x}_i$ to get $\mathbf{x}_i^{(N)}$, $i = 1, 2, \cdots, n$};
Let $X^{(N)} = \{\mathbf{x}_i^{(N)}: i = 1, 2, \cdots, n\}$;
Select a random sample $X^{(S)}$, of size n_s by SRSWOR-scheme from $X^{(N)}$ using χ^2 or divergence or select a random sample of size $n_s = n/2$;
Run Sammon_projection with $X^{(S)}$ to get $Y^{(S)} = \{\mathbf{y}_i^{(S)} \in \mathcal{R}^q: i = 1, 2, \cdots, n_s\}$ where $\mathbf{y}_i^{(S)}$ corresponds to $\mathbf{x}_i^{(S)}$;
Normalize $\mathbf{y}_i^{(S)}$ to get $\mathbf{y}_i^{(N)}$, $i = 1, 2, \cdots, n_s$};
Let $Y^{(N)} = \{\mathbf{y}_i^{(N)}: i = 1, 2, \cdots, n_s\}$;
Train an MLP with $X^{(S)}$ and $Y^{(N)}$, $\mathbf{y}_i^{(N)}$ is the target corresponding to $\mathbf{x}_i^{(S)}$;
Use this trained MLP to project the complete data set $X^{(N)}$ and any new data points.
}

B. Method 2

In Method 1, we used 50% sample, although 50% sample is usually not needed. A small but adequate representation

---

Kohonen's_SOFM( )
{
Input $X = \{\mathbf{x}_i \in \mathcal{R}^p : i = 1, 2, \cdots, n\}$;
Input m; /* The display grid size, a square of $m \times m$ lattice is assumed */
Input maxstep; /* Maximum number of updating steps; */
Input Nd_o; /* Initial neighborhood size */
Input α_o; /* The Initial step size (learning coefficient) */
Input σ_o and σ_f; /* Parameters to control effective step size */
Generate initial weight vectors randomly
$\{\mathbf{w}_{ij}, i = 1, 2, \cdots, m; j = 1, 2, \cdots, m\}$;
$t = 0$;
while($t <$ maxstep)
{
Select $\mathbf{x}$ randomly from X;
Find $r = \min_i\{||\mathbf{x} - \mathbf{w}_i||\}$;
/* r & i stand for 2-D indices that uniquely identify a node on the display lattice; */
$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha_t * g_t(dist(r, i)) * (\mathbf{x} - \mathbf{w}_i(t))\ \forall i \in Nd_t(r)$;
$\mathbf{w}_i(t+1) = \mathbf{w}_i(t)\ \forall i \notin Nd_t(r)$;
where $g_t(d) = e^{-d^2/\sigma_t^2}$;
$t = t + 1$;

$$\alpha_t = \alpha_o * \left(1 - \frac{t}{maxstep}\right);$$

$$Nd_t = Nd_o - \frac{t * (Nd_o - 1)}{maxstep};$$

$$\sigma_t = \sigma_o - \frac{t * (\sigma_o - \sigma_f)}{maxstep};$$

}
}

of the data set X should be enough. We also explained how an adequate sample can be selected [28]. Here we propose a hybrid network-based scheme which uses both Kohonen's self organizing feature map (SOFM) and MLP. We use SOFM for finding a small representative data set. Here the objective is to reduce the size of the data set further. Thus, instead of using samples that retain the structure of the original data, we intend to use one of few prototypes which encode a small region in the feature space. Note that, even when the data set does not have a very well-defined cluster substructure, such encoding will be there.

The set of vectors encoding the data set can be obtained, if we can cluster it by a suitable clustering algorithm with the *right number* of clusters. But choosing the right clustering algorithm and the right number of clusters are difficult problems. When clustering of the data is the only problem to be solved, we use some clustering algorithm to extract the natural substructures present in the data. For example, in the IRIS data [31] one would try to find three clusters corresponding to three actual types of flowers. But for the problem at hand, projecting only the centroids of the three clusters will not suffice. Rather, here for each cluster we need to extract a few prototypes each of which locally encodes a small but reasonable area of the feature space and all of them taken together encode the entire cluster. Use of SOFM [29], [30] could be a good solution here as we can start with a reasonably large number of prototypes and then use the weight vectors associated with winner cells of an SOFM trained with the set X. This can be done because in SOFM, each cell in the competitive layer has a tendency to encode (become a prototype of) a region in the input space according to the density distribution of the data. This is expected to reduce the sample size significantly. Before describing the proposed algorithm, for completeness, we briefly present the SOFM algorithm.

SOFM: SOFM [29], [30] is a transformation from $\mathcal{R}^p \rightarrow \mathcal{R}^2$ that is often advocated for visualization of metric-topological relationships and distributional density properties of feature vectors (signals) X in $\mathcal{R}^p$. The visual display produced by SOFM presumably helps one form a hypothesis about topological structures in X. In principle, X can be transformed onto a display lattice in $\mathcal{R}^q$ for any q; in practice, visual displays can be made only for $q \leq 3$, and are usually made on a linear or planar configuration arranged as a rectangular or hexagonal lattice. In this work we use only square $(m \times m)$ displays.

Input vectors $\mathbf{x} \in \mathcal{R}^p$ are distributed by a fan-out layer to each of the $(m \times m)$ output nodes in the competitive layer. Each node in this layer has a weight vector $\mathbf{w}_{ij} \in \mathcal{R}^p$ attached to it. We let $O_p = \{\mathbf{w}_{ij}\} \subset \mathcal{R}^p$ denote the set of m^2 weight vectors.

SOFM usually begins with random initialization of the weight vectors $\mathbf{w}_{ij}$. To simplify notation we suppress double subscripts. Now let $\mathbf{x} \in \mathcal{R}^p$ enter the network and let t denote the current iteration number. Find $\mathbf{w}_r$, the vector in O_p that best matches $\mathbf{x}$ in the sense of minimum Euclidian distance in $\mathcal{R}^p$. This vector has a (logical) "image" which is the cell in the competitive layer with subscript r. Next, a topological (spatial) neighborhood $N_t(r)$ centered at r is defined. Finally, $\mathbf{w}_r$ and the other weight vectors associated with cells in the spatial neighborhood $N_t(r)$ are updated using the rule

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + h_{ri}(t)(\mathbf{x} - \mathbf{w}_i(t)). \tag{18}$$

Here r is the index of the "winner" prototype,

$$r = \min_i\{\|\mathbf{x} - \mathbf{w}_i\|\}, \text{ and } \|*\| \text{ is the Euclidean norm on } \mathcal{R}^p. \tag{19}$$

The function $h_{ri}(t)$ in (18) is used to express the strength of interaction between cells r and i in the competitive layer. Usually $h_{ri}(t)$ decreases with t, and for a fixed t it decreases as the distance from cell i to cell r increases. A common choice for $h_{ri}(t) = \alpha_t * e^{-\text{dist}^2(r,i)/\sigma_t^2}$, where α_t and σ_t decrease with time t, and dist(r, i) is the Euclidian distance between nodes r and i on the two-dimensional lattice. The topological neighborhood $N_t(r)$ also decreases with time.

Our implementation of Kohonen's network algorithm is given as shown at the bottom of the previous page.

The algorithm for Method 2 can now be summarized as follows:

Method-2( )
{
Input $X = \{\mathbf{x}_i \in \mathcal{R}^p$: $i = 1, 2, \cdots, n\}$;

/* **Use Kohonen's_SOFM Algorithm** */

Run Kohonen's_SOFM with X as inputs;
Generate $\tilde{X} = \{\mathbf{w}_i \in \mathcal{R}^p$: *i is a winner cell*$\}$;
Let there be $\hat{n}$ such winners;
Normalize $(X \cup \tilde{X})$, to get $\mathbf{x}_i^{(N)}$, $i = 1, 2, \cdots, n$ and $\mathbf{w}_i$ to get $\mathbf{w}_i^{(N)}$, $i = 1, 2, \cdots, \hat{n}$;
Let $X^{(N)} = \{\mathbf{x}_i^{(N)}$: $i = 1, 2, \cdots, n\}$ and $\tilde{X}^{(N)} = \{\mathbf{w}_i^{(N)}$: $i = 1, 2, \cdots, \hat{n}\}$;

/* **Use Sammon's Projection Method** */

Run Sammon's_projection with $\tilde{X}^{(N)}$ to get
$Y = \{\mathbf{y}_i \in \mathcal{R}^q$: $i = 1, 2, \cdots, \hat{n}\}$
where $\mathbf{y}_i$ corresponds to $\mathbf{w}_i^{(N)}$;
Normalize $\mathbf{y}_i$ to get $\mathbf{y}_i^{(N)}$, $i = 1, 2, \cdots, \hat{n}$;
Let $Y^{(N)} = \{\mathbf{y}_i^{(N)}$: $i = 1, 2, \cdots, \hat{n}\}$;

/* **Use Backpropagation Algorithm** */

Train an MLP with $\tilde{X}^{(N)}$ as inputs and $Y^{(N)}$ as targets
$\mathbf{y}_i^{(N)}$ is the target corresponding to $\mathbf{w}_i^{(N)}$;

Use this trained MLP to project the complete set $X^{(N)}$ and new data points.
}

V. Implementation and Results

A. Data Sets

To demonstrate the effectiveness of the proposed schemes we implemented both of them as well as Sammon's algorithm and Sammon_Network of Jain and Mao. All of these four algorithms are tested on three data sets named **Iris**, **Sphere-Shell**, and **Elongated-Clusters**.

Iris [31] is a well-known data set consisting 150 points from three classes in a four-dimensional space. Each class has 50 points. One of the classes is well separated from the rest while the other two have some overlap.

Sphere-Shell [32] is a synthetic data set consisting of 1000 points in three dimensions. 500 points are selected randomly within a hemisphere of radius $r1$ (=0.6) and rest 500 from a shell defined by two hemispheres of radius $r2$ (=2.0), and $r3$ (=2.013), such that $r1 < r2 < r3$.

Elongated clusters [9] is also a synthetic data set consisting of 2 elongated clusters of 500 points each in three-space.

B. Computational Protocols

Here we list the values of different parameters used in our implementations. These are kept the same irrespective of the data set used.

1) **PCA_Network:**
   $\epsilon = 0.000\,01$
   $maxstep = 5000$ iterations
   $\eta = 1.5$
   $\mu = 1.5$
   $\beta = 0.1$
   $\alpha = 0.999$
2) **Sammon_Projection:**
   $\epsilon = 0.0001$
   $maxstep = 100$ epochs
   $\alpha = 0.5$
3) **Sammon_Network:**
   $\epsilon = 0.000\,01$
   $maxstep = 1000$ epochs
   $\alpha = 0.9$
   $\beta = 0.7$
   Hidden Layers $= 1$
   Hidden Nodes $= 30$
4) **Backpropagation:**
   $\epsilon = 0.001$
   $maxstep = 1000$ epochs
   $\eta = 0.9$
   $\alpha = 0.7$
   Hidden Layers $= 1$
   Hidden Nodes $= 30$
5) **Kohonen's_SOFM:**
   $m = 25$
   $maxstep = 1000$ iterations
   $Nd_0 = 21$
   $\alpha_0 = 0.9$
   $\sigma_0 = 4.0$
   $\sigma_f = 0.5$

TABLE I
Sample Size Used in Different Methods

Data Set	Data size	Sammon's Projection	Sammon's Net	Method 1	Method 2
IRIS	150	150	75	75	43
Elongated Clusters	1000	1000	500	500	68
Sphere-Shell	1000	1000	500	500	145

TABLE II
The CPU Time (Secs) for Various Methods

Data Set	Sammon_Projection	Sammon_Network	Method 1	Method 2
IRIS	205.61	1288.53	588.99	447.25
Elongated Clusters	9094.53	7555.75	5390.94	502.36
Sphere-Shell	8995.78	7611.33	5508.58	1218.30

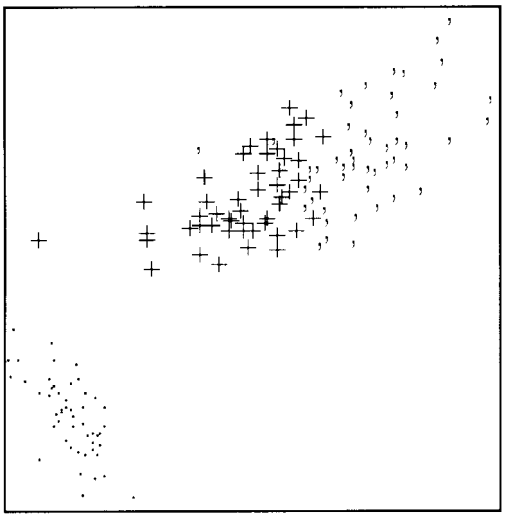

Fig. 1. Iris—Projection by Sammon's Method.

C. Results

Table I lists the sample size used by different methods for different data sets. Column 6 in Table I shows that SOFM-based selection can reduce the sample size drastically, of course, at the cost of training of SOFM. It is interesting to note that for the elongated cluster, the number of required data points as suggested by SOFM is less than 7% of the size of the original data set.

Table II shows the CPU time needed by different methods for the three data sets. For Iris, since it is a very small data set, Sammon-Network required about six times CPU time as that of Sammon's algorithm; while the proposed methods required about 2.5× time as that of Sammon's algorithm. In fact for any small data set none of the network implementations is expected to be beneficial. Figs. 1–4 display the scatterplots of the two-dimensional data sets projected by the four methods for Iris.

For the elongated clusters Jain and Mao's method reduces the CPU time by about 17% while the reduction for Method

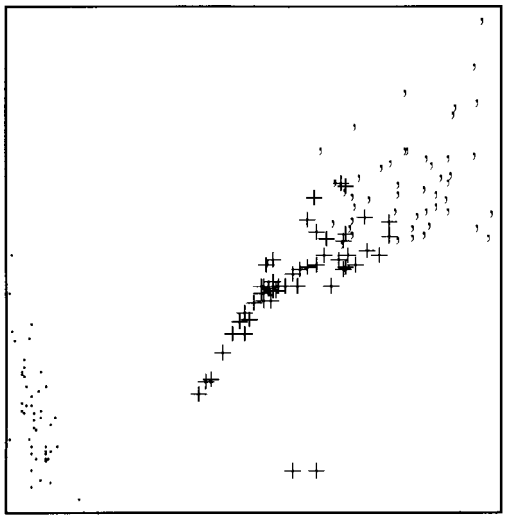

Fig. 2. Iris—Projection by Method 1.

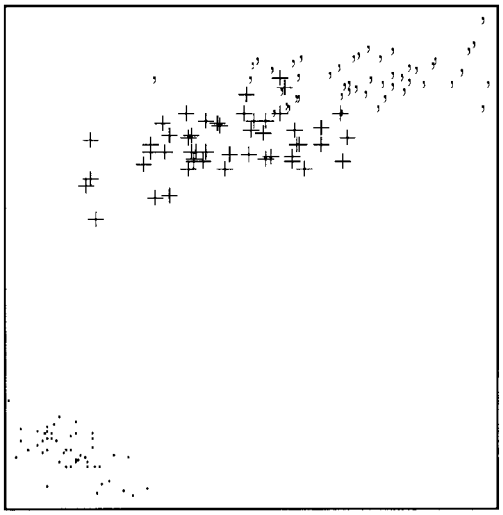

Fig. 3. Iris—Projection by Sammon_Network.

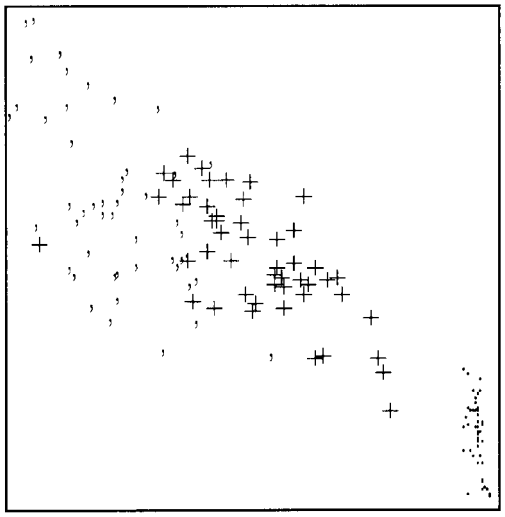

Fig. 4. Iris—Projection by Method 2.

TABLE III
SAMMON'S ERROR FOR VARIOUS METHODS

Data Set	Sammon_Projection	Sammon_Network	Method 1	Method 2
IRIS	0.006 590	0.012 521	0.061 734	0.042 415
Elongated Clusters	0.029 305	0.341 532	0.344 348	0.345 785
Sphere-Shell	0.000 889	0.09	0.03	0.05

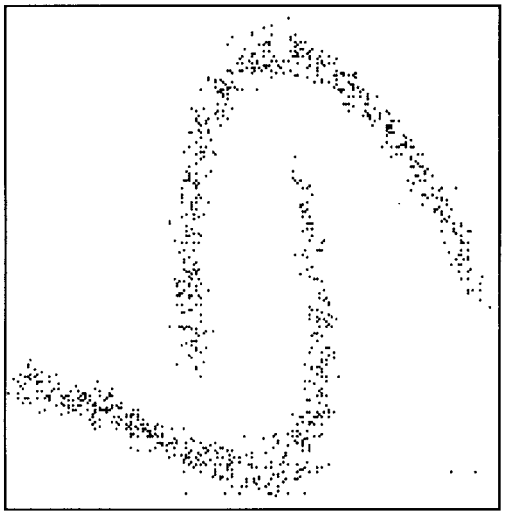

Fig. 5. Elongated clusters—Projection by Sammon's method.

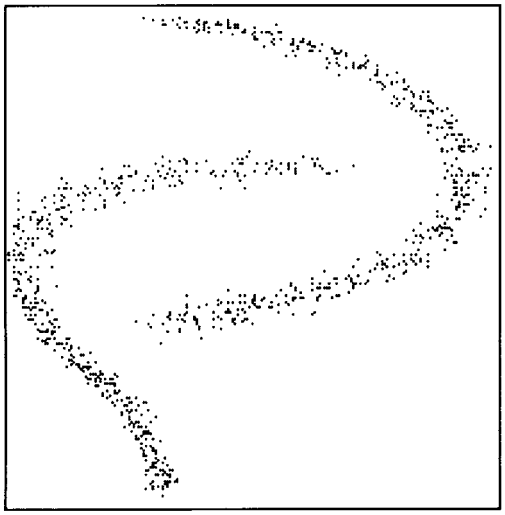

Fig. 6. Elongated clusters—Projection by Method 1.

1 and Method 2 are nearly 40 and 95%. The computation time for Method 2 is about 7% of Jain and Mao's method, yet the performance of the proposed methods and Sammon_Network are quite comparable in terms of Sammon's error function (Table III). Visually also they are quite comparable, see Figs. 5–8.

Finally for the Sphere-Shell data also a significant improvement in computation time is exhibited by the proposed methods over both Sammon_Network and Sammon's algorithm. It is interesting to note that Sammon's error for the

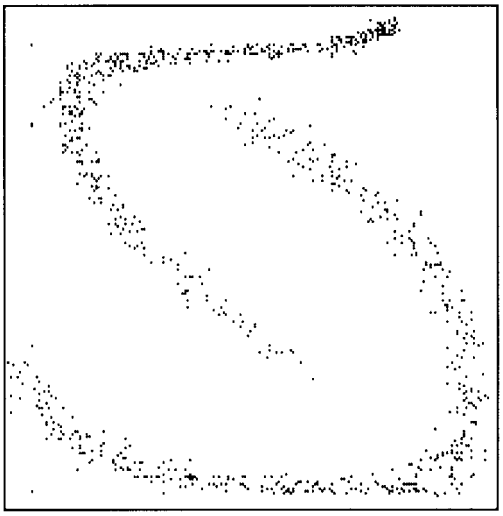

Fig. 7. Elongated clusters—Projection by Sammon_Network.

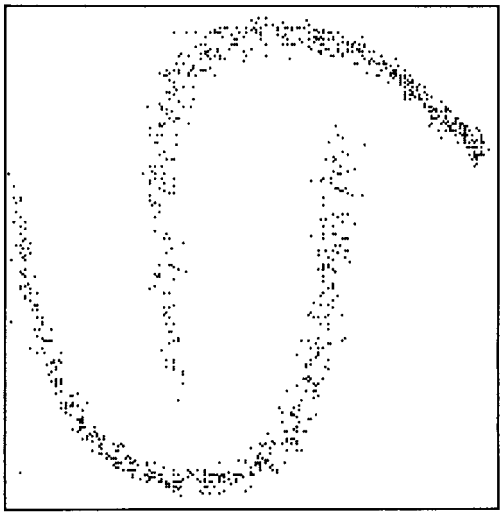

Fig. 8. Elongated clusters—Projection by Method 2.

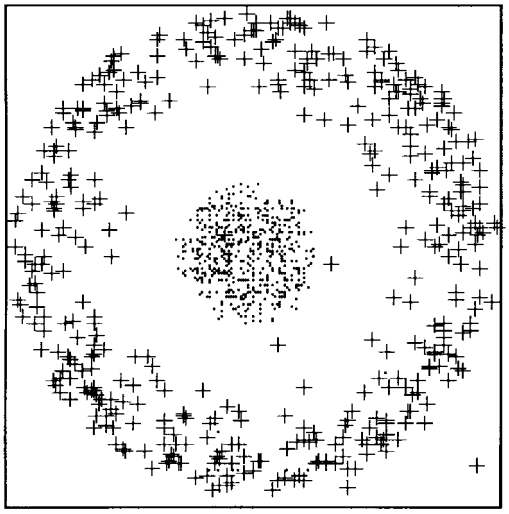

Fig. 9. Sphere-shell—Projection by Sammon's method.

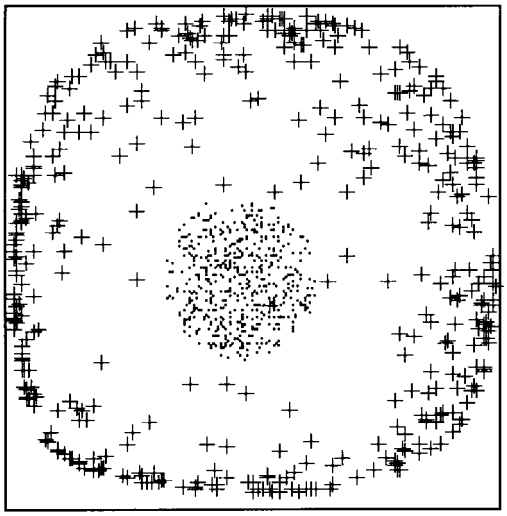

Fig. 10. Sphere-mtbdshell—Projection by Method 1.

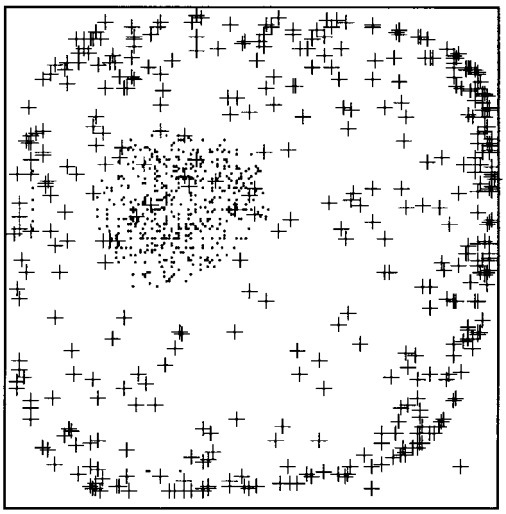

Fig. 11. Sphere-shell—Projection by Sammon_Network.

Sphere–Shell data is even less than that for the other two methods. For elongated clusters and Sphere–Shell, we used the same sample size and for them each algorithm required CPU time of the same order, but the Sammon's error for Sphere–Shell is much smaller than that for elongated clusters. There could be several reasons for this; for example, the Sammon's error surface for elongated clusters could be more complex with many local minima than that for Sphere–Shell and the initiliazations used may be such that the search got stuck in one such bad minima. The scatterplots of the projected data for sphere-shell in Figs. 9–12 show that of the three NN implementations Method 1 produces the best result.

Summarizing results we offer the following conclusions: for small data sets Sammon's algorithm may be computationally better but it does not have the prediction capability. For large data sets, the computational overhead is likely to follow the ordering: Sammon's algorithm $>$ Sammon_Network $>$ Method 1 $>$ Method 2. The predictability of all the three network based methods are quite comparable in terms of

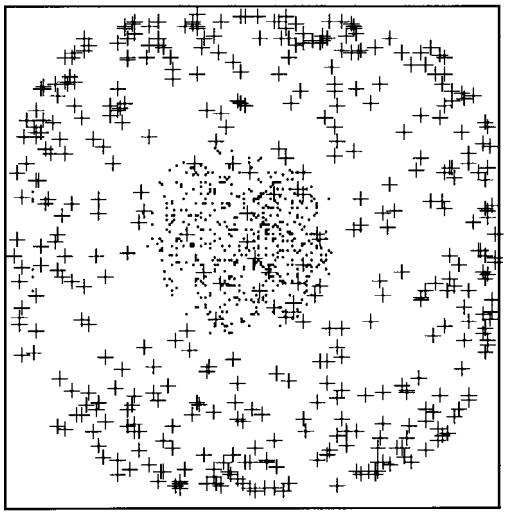

Fig. 12. Sphere-shell—Projection by Method 2.

Sammon's error. But scatter plots of the projected data sets show that for nonlinear data the proposed methods are better than Jain and Mao's method.

VI. Conclusion

We have proposed two connectionist schemes for structure preserving dimensionality reduction (feature extraction). The first method, Method 1, unlike Jain and Mao's method which attempts to minimize Sammon's error, uses Sammon's algorithm to project a small data set $X^{(S)}$ selected based on statistical criterion that ensures preservation of the distributional characteristics of the original data set. Then an MLP is used to capture the relation between $X^{(S)}$ and its Sammon projected values. This trained MLP is then used for further projection of new data points. The second method, Method 2, exploits the substructure encoding characteristic of Kohonen's SOFM and the function approximation capability of MLP's. Both methods drastically reduce the computation time compared to Sammon's algorithm and Sammon_Network of Jain and Mao, and exhibit good prediction capability for new data points. Method 2 reduces the size of the data set drastically and hence the quality of the projected map is not as good as that of Method 1. Instead of considering only the winner nodes, it may be interesting to take all prototypes of the SOFM. In this case the number of nodes on the display lattice should be taken much smaller than the data set size. This will ensure reduction of data set to be used with Sammon's algorithm and at the same time consideration of the nonwinner nodes is expected to result in a better projection map. To summarize the proposed methods are conceptually sound and simple and overcome the problems of both Sammon's method and the network implementation of Jain and Mao.

References

[1] K. Fukunga, *Introduction to Statistical Pattern Recognition*, 2nd ed. New York: Academic, 1990.

[2] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[3] J. W. Sammon, Jr., "A nonlinear mapping for data structure analysis," *IEEE Trans. Comput.*, vol. C-18, pp. 401–409, 1969.

[4] B. Schachter, "A nonlinear mapping algorithm for large databases," *Comput. Graphics Image Process.*, vol. 7, pp. 271–278, 1978.

[5] C. E. Pykett, "Improving the efficiency of Sammon's nonlinear mapping by using clustering archetypes," *Electron. Lett.*, vol. 14, pp. 799–800, 1980.

[6] C. L. Chang and R. C. T. Lee, "A heuristic relaxation method for nonlinear mapping in cluster analysis," *IEEE Trans. Syst., Man., Cybern.*, vol. SMC-3, pp. 197–200, 1973.

[7] G. Biswas, A. K. Jain, and R. C. Dubes, "Evaluation of projection algorithms," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-3, pp. 701–708.

[8] A. K. Jain and J. Mao, "Artificial neural networks for nonlinear projection of multivariate data," in *Proc. IEEE Int. Joint Conf. on Neural Networks*, vol. 3, pp. 59–69, 1992.

[9] J. Mao and A. K. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Trans. Neural Networks*, vol. 6, no. 2, pp. 296–317, 1995.

[10] E. Oja, "A simplified neuron model as a principal component analyzer," *J. Math Biol.*, vol. 15, pp. 267–273, 1982.

[11] ——, "Neural networks, principal components, and subspaces," *Int. J. Neural Syst.*, vol. 1, pp. 61–68, 1989.

[12] J. Rubner and K. Schulten, "Development of feature detectors by self organization," *Biol. Cybern.*, vol. 62, pp. 193–199, 1990.

[13] J. Rubner and P. Tavan, "A self-organizing network for principal component analysis," *Europhys. Lett.*, vol. 10, pp. 693–698, 1989.

[14] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural networks," *Neural Networks*, vol. 2, pp. 459–473, 1989.

[15] H. M. Abbas and M. M. Fahmy, "A neural model for adaptive Karhunen–Loeve Transform (KLT)," in *Proc. IEEE Int. Joint Conf. Neural Networks*, Baltimore, MD, June 1992, vol. 2, pp. 975–980.

[16] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural Networks*, vol. 1, pp. 53–58, 1989.

[17] P. Foldiak, "Adaptive network for optimal linear feature extraction," in *Proc. IEEE Int. Joint Conf. Neural Networks*, Washington, DC, 1989, vol. 1, pp. 401–405.

[18] K. Hornik and C. M. Kuan, "Convergence analysis of local feature extraction algorithm," *Neural Networks*, vol. 5, pp. 229–240, 1992.

[19] H. Kuhnel and P. Tavan, "A network for discriminant analysis," in *Artificial Neural Networks*, T. Kohonen, K. Makisara, O. Simula, and J. Kangas, Eds. New York: Elsevier, 1991.

[20] R. Linsker, "From basic network principles to neural architecture," in *Proc. Nat. Academy Sci.*, vol. 83, 1986, pp. 7508–7512, 8390–8394, 9779–9783.

[21] E. Oja and J. Karhunen, "Nonlinear PCA: Algorithms and applications," Report A18, Espoo, Finland: Helsinki Univ. Technol., Lab. Comput. Inform. Sci., Sept. 1993.

[22] J. Karhunen and J. Joutsensalo, "Representation and separation of signals using nonlinear PCA type learning," *Neural Networks*, vol. 7, pp. 113–127.

[23] P. Comon, "Independent component analysis—A new concept?," *Signal Processing*, vol. 36, no. 3, pp. 287–314, 1994.

[24] R. Hetcht-Nielsen, "Theory of backpropagation neural network," in *Proc. IJCNN'89*, Washington, D.C., vol. I, June 1989, pp. 593–605.

[25] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366.

[26] G. Cybenko, "Continuous valued neural networks with two hidden layers are sufficient," *Math. Contr., Signals, Syst,*, vol. 2, pp. 303–314, 1985.

[27] E. K. Blum and L. K. Li, "Approximation theory and feedforward networks," *Neural Networks*, vol. 4, pp. 511–515, 1991.

[28] B. Uma Shankar and N. R. Pal, "FFCM: An effective approach for large data sets, in *Proc. 3rd Int. Conf. Fuzzy Logic, Neural Nets and Soft Computing*, Iizuka, Japan, 1994, pp. 331–332.

[29] T. Kononen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, pp. 59–69, 1982.

[30] ——, "Self-organized neural network," *Proc. IEEE*, vol. 43, pp. 59–69, 1990.

[31] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, pp. 178–188, 1936.

[32] H. Niemann, "Linear and nonlinear mapping of patterns," *Pattern Recognition*, vol. 12, pp. 83–87.

Nikhil R. Pal (M'91) received the B.Sc. degree in physics and master's degree in business management from the University of Calcutta, India, in 1979 and 1982, respectively. He received the M.Tech. and Ph.D. degrees, both in computer science, from the Indian Statistical Institute in 1984 and 1991, respectively.

Currently, he is a Professor in the Machine Intelligence Unit of Indian Statistical Institute, Calcutta. From September 1991 to February 1993 and July 1994 to December 1994, he was with the Computer Science Department of the University of West Florida, Pensacola. He was a Guest Faculty Member of the University of Calcutta also. His research interests include image processing, pattern recognition, fuzzy sets theory, measures of uncertainty, neural networks, genetic algorithms, and fuzzy logic controllers.

Dr. Pal is an Associate Editor of the *International Journal of Approximate Reasoning* and IEEE TRANSACTIONS ON FUZZY SYSTEMS. He has acted as a Guest Editor of different special issues of *International Journal of Approximate Reasoning*, *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems*, and *Fuzzy Sets and Systems*.

Vijay Kumar Eluri received the B.Tech. degree in computer science from Andhra University, Visakhapatnam, India, in 1993 and the M. Tech. degree in computer science from the Indian Statistical Institute, Calcutta, in 1996.

Currently, he is working as a Software Design Engineer in Vedika International Pvt. Ltd., Calcutta. His research interest includes neural networks, fuzzy logic, dimensionality reduction, and pattern recognition.