Learning Approximate Isometries using Multi-Dimensional Scaling based on Radial Basis Function Neural Networks

Ryan Gonet[a], RuiZhi Yu[b],
Mingbo Ma[c] and Georgios C. Anagnostopoulos[c]

[a]Computer Science, University of South Florida, Tampa, Florida;
[b]Electrical Engineering, Princeton University, Princeton, New Jersey;
[c]Electrical & Computer Engineering, Florida Institute of Technology, Melbourne, Florida

ABSTRACT

This paper presents a new *multi-dimensional scaling (MDS)* technique for generating approximate isometries by way of using *Radial Basis Function neural networks (RBF-NN)*. Metric MDS using the Sammon mapping[1] is available and commonly used to create approximate isometries in order to perform data visualization. However, classical-Sammon mapping has the disadvantage of being unable to interpolate or extrapolate novel samples. Other techniques, such as one using *Multi-Layer Perceptron (MLP)* to implement the mapping, cannot map datasets where only dissimilarities between patterns are known. By using RBF-NNs to train a model to map patterns from a higher dimension into a lower dimension, one can both perform dimensionality reduction on datasets consisting only of dissimilarities and also interpolate and extrapolate novel patterns. For training, both the delta rule, through conjugate gradient descent and the limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton method were used. Step lengths were calculated using a line search that finds step lengths obeying the strong Wolfe conditions. Experimental results using a prototype implementation of RBF-NNs to perform Sammon mapping demonstrate that approximate isometries can be found. They also show that datasets consisting of only dissimilarities can still be visualized, where the reduction shows a structural relationship between patterns. This new ability, in conjunction with the ability to interpolate and extrapolate previously unseen patterns, creates novel opportunities for the application of MDS using Sammon mapping as a visualization technique.

Keywords: Multi-Dimensional Scaling, Sammon Map, Radial Basis Function Neural Networks, Dimensionality-Reduction, Data Visualization

1 INTRODUCTION

Multi-Dimensional Scaling (MDS) is a common class of techniques for performing distance preserving dimensional reduction on datasets with dimensionality greater than three. There are two types of MDS: metric MDS and non-metric MDS. Metric MDS attempts to preserve distances when some metric is used to measure distance, while non-metric MDS attemps to preserve the rank order of distances and relative dissimilarities. In this paper, we will be dealing with metric MDS. One of the most known methods for performing metric MDS, *Sammon's Non-Linear Mapping (SNLM)*, is frequently used in order to visualize the dataset in lower dimensions. By preserving distances while projecting into a lower dimensional space with SNLM, the similarities and dissimilarities between patterns may be readily visible, if the comparison is based on some distance measure. SNLM is useful for visualizing data in various areas of research such as data mining and psychology, where data may be multi-dimensional and/or not purely numeric. One of its drawbacks, however, is that it cannot interpolate or extrapolate from new patterns without re-running the training algorithm, which may take too long for the certain circumstances it could be useful in.

Further author information: (Send correspondence to G.C.A.)
G.C.A.: E-mail: georgio@fit.edu, Telephone: 1 321 674 7125
R.G.: E-mail: rgonet@mail.usf.edu
R.Y.: E-mail: ruizhiyu@princeton.edu
M.M.: E-mail: mma2008@fit.edu

Historically, MDS was developed by Young and Householder in 1938[2] and further developed by Torgerson[3] in 1952. The SNLM technique was developed by Sammon in 1969[1]. While other mapping-based metric MDS techniques, such as the one based on *Multi-Layer Perceptron (MLP)*[4], can be used to perform Sammon mapping while achieving the ability to interpolate new patterns, they lose their usefulness when dealing with datasets that have only the pair-wise dissimilarities available.

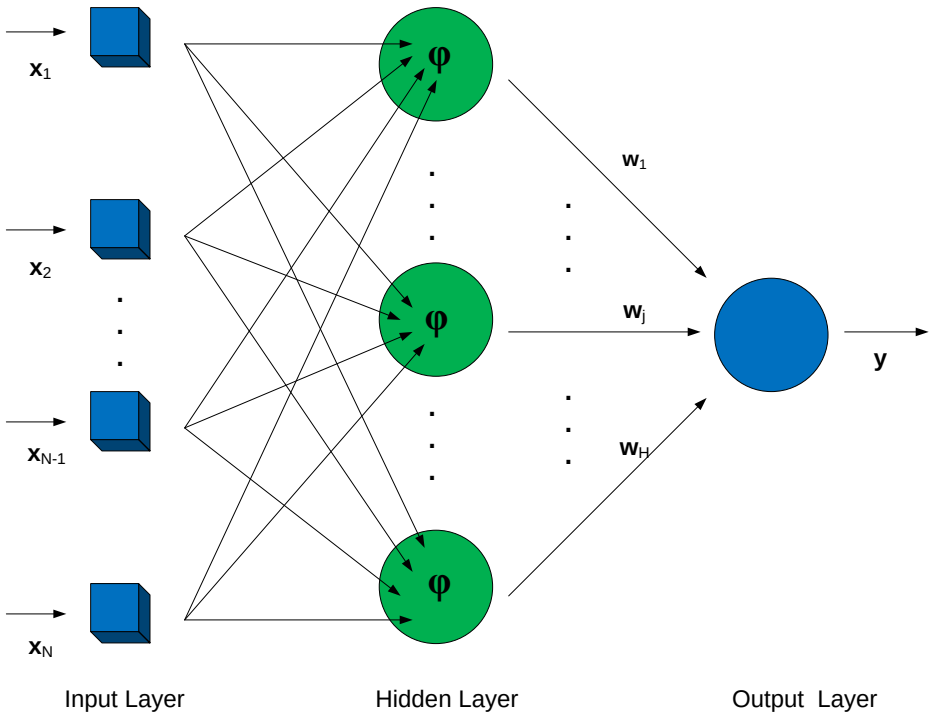

Figure 1. RBF-SNLM. The input is mapped from the original (possibly high-dimensional) feature space to an intermediate hidden layer via *Radial Basis Functions (RBFs)* before being mapped to the target space, which is typically 2- or 3-dimensional.

To address these deficiencies, we have developed a technique for performing SNLM using *Radial-basis function neural networks (RBF-NN)*, which we abbreviate as *Radial-basis function Sammon non-linear mapping (RBF-SNLM)*. Each network learns a parameterized, isometric mapping from possibly high dimension input patterns to a lower dimensional space that can be visualized (in 2 or 3 dimensions). The structure of an RBF-NN is shown in Fig. 1.

Using *Radial basis functions (RBFs)* in interpolation models was studied by Powell in 1985[5] and then Light in 1993[6]. RBF-NNs were proven to be universal approximators by Park and Sandberg in 1991[7]. Using RBF-NNs to perform Sammon mapping allows for the ability to interpolate and extrapolate even for problems which only offer input pattern dissimilarities rather than the exact sample values.

The scope of our research is first to develop the algorithm to train a RBF-NN to perform SNLM. We also need to investigate ways of improving performance in convergence rate, memory use, and execution time. Additionally, we aim to showcase the potential utility of RBF-SNLM by applying it to a select group of artificial and real-life visualization problems. Our methodology consists of finding different types of datasets and finding methods of applying RBF-NNs to them in order to create a Sammon mapping to an arbitrary error.

In this document, we present experiments performed on datasets using RBF-NNs. The results of our experimentation show promise that, if proper methods are used to avoid local minima, RBF-NNs can train a model to perform SNLM to an arbitrary error.

This document is organized in the following manner. Section 2 is a discussion of MDS where we elaborate more thoroughly on SNLM and the fundamental concepts behind RBF-SNLM. In section 3, we describe our algorithm in detail and present ways of decreasing the execution time required by RBF-SNLM. Section 4 contains the experimental results where we describe our experiments using RBF-SNLM. There,we show our results and analyze them. In that section, we also discuss methods for improving the quality of interpolation. Finally, in section 5 we present our concluding remarks and suggest future work.

For the remainder for this document, we will denote N as the number of patterns in the training set, D as the number of dimensions of the input feature space, M as the number of dimensions of the output feature space, and H as the number of hidden units in the single hidden layer.

2 MULTI-DIMENSIONAL SCALING

2.1 Background Information

MDS is a set of statistical techniques that aims to reduce datasets from a possibly high dimensional space to a lower one, usually 2D or 3D, for visualization purposes. It aims to preserve, as much as possible, the similarities and dissimilarities of the patterns in the original higher dimensional data space to the mapping in the low dimensionality target space[8].

The simplest form of MDS, which we refer to as *Classical-Sammon Non-Linear Mapping (Classical-SNLM)* is simply a minimization problem where the pattern locations are randomly initialized in the target mapped space and adjusted accordingly until the error function is minimized. The error function in Classical-SNLM is determined by the sum of the differences in the dissimilarities between each pair of training patterns in the source space and target space. Frequently, some type of gradient of the parameters with respect to the error function is calculated and a line search method is used in the minimization to find an appropriate step length.

Since its development, improvements have been made to the Classical-SNLM method. One of the key disadvantages of Classical-SNLM is the lack of interpolation/extrapolation ability. That is, given an input pattern not used in designing the SNLM map, it is impossible to know the image of this particular pattern in the lower dimension target space.

To overcome this constraint, researchers have used a variety of non-linear regression models to implement SNLM. Among these are the multi-layer feedforward networks. One of the most celebrated of them is the MLP[9,10,11]. The response of these models can be expressed by

$$\mathbf{y} = \mathbf{F}(\mathbf{x}, \theta), \tag{2.1}$$

where $\mathbf{x}$ is an input pattern, $\mathbf{y}$ is its image in the target space, and θ is a vector of all the parameters in the model. If one were to use an MLP to find a Sammon mapping, the difference between it and Classical-SNLM would be that, rather than directly optimizing $\mathbf{y}$ in the minimization problem, the model parameters θ are optimized instead.

For MLP neural networks, it has been shown by Hornik, Stinchcombe, and White[12] that "standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired accuracy, provided sufficiently many hidden units are available."

The number of units in each hidden layer of an MLP depends upon the specific function that is being approximated. While there is no known general formula for the number of hidden units required per layer to approximate a function with arbitrary accuracy[13], for function spaces, there may exist a formula for the number of necessary hidden units, such as for the set of boolean functions. Generally, the number of hidden units is determined through experimentation.

A disadvantage of using an MLP to find a Sammon mapping is that the precise coordinates of the points have to be known. If only the distances (or, in general, dissimilarities) are known, then MLP cannot be used to find the mapping. For this reason, we look at another multi-layer feedforward neural network, the RBF-NN. They

differ from MLPs in that they only have a single hidden layer. This hidden layer is non-linear in that it uses a vector of RBFs

$$\boldsymbol{\phi}(\mathbf{x}_n) = [\phi_1(\mathbf{x}_n), \phi_2(\mathbf{x}_n), \ldots, \phi_H(\mathbf{x}_n)]^T \tag{2.2}$$

to map the input patterns to a higher dimension. The RBFs are non-linear kernel activation functions that depend on the distances from the centers $\mathbf{C} \in \mathbb{R}^{H \times M}$ to the input patterns $\mathbf{x} \in \mathbb{R}^{M \times 1}$. The RBFs must depend on these distances but can be any kernel function of the form

$$\boldsymbol{\phi}_h(\mathbf{x}_n) = \phi_h(\|\mathbf{x_n} - \mathbf{c}_h\|) \qquad h = 1, \ldots, H; n = 1, \ldots, N \tag{2.3}$$

where $\mathbf{x}$ is an input pattern and $\mathbf{c}_h$ is the vector of the centers of the h^{th} RBF. The *interpolation matrix*

$$\boldsymbol{\Phi} \hat{=} \begin{bmatrix} \phi_1(\mathbf{x}_1, s_1, \mathbf{c_1}) & \phi_1(\mathbf{x}_2, s_1, \mathbf{c_1}) & \cdots & \phi_1(\mathbf{x}_N, s_1, \mathbf{c_1}) \\ \phi_2(\mathbf{x}_1, s_2, \mathbf{c_2}) & \phi_2(\mathbf{x}_2, s_2, \mathbf{c_2}) & \cdots & \phi_2(\mathbf{x}_N, s_2, \mathbf{c_2}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_H(\mathbf{x}_1, s_H, \mathbf{c_H}) & \phi_H(\mathbf{x}_2, s_H, \mathbf{c_H}) & \cdots & \phi_H(\mathbf{x}_N, s_H, \mathbf{c_H}) \end{bmatrix}. \tag{2.4}$$

contains all RBFs evaluated for each hidden node and pattern pair. Furthermore, Micchelli's Theorem[14] demonstrates that if interpolation is desired, the interpolation matrix must be nonsingular. A common function to use as a RBF is the Gaussian function

$$\phi_h(\mathbf{x}, \mathbf{c}_h, s_h) = exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_h\|^2}{s_h}\right) \qquad h = 1, \ldots, H \tag{2.5}$$

where s_h is the spread parameter that determines the width of the Gaussian function ϕ_h. The number of RBFs, H, eventually influences the accuracy of the network.

The output layer of an RBF neural network is linear and the network's response is defined as

$$\mathbf{y} = \boldsymbol{\Phi}(\mathbf{x}, \boldsymbol{\Theta})^T \mathbf{W} \tag{2.6}$$

where $\boldsymbol{\Theta}$ consolidates all network parameters.

3 PROPOSED APPROACH

3.1 Radial-basis function Sammon non-linear mapping (RBF-SNLM)

Our technique of performing dimensionality reduction for data visualization trains an RBF-NN to approximate a mapping from a dataset of a possibly large number of dimensions to a space of either one, two, or three dimensions. The rationale for choosing RBF-NNs lies in the fact that they can interpolate/extrapolate patterns that have not been used to design the mapping (i.e. other than training patterns). Also, in contrast to MLP, the network's use of RBFs to map the input patterns to the hidden layer outputs allows for the usage of datasets where only distances between patterns are known or can be derived. Another advantage is that the model is easy to train due to there being only one hidden layer.

We name this new technique RBF-SNLM. The points from the original database are non-linearly mapped via the RBF functions, with each of the H RBF functions defined by

$$\phi_h(\mathbf{x}, \mathbf{c}_h, s_h) = exp\left(-\frac{\delta^2(\mathbf{x}, \mathbf{c}_h)}{s_h}\right) \tag{3.1}$$

where $\delta(\cdot)$ is a dissimilarity metric. The change in Eq. 2.5 from the vector norm to a more abstract dissimilarity in Eq. 3.1 is to emphasize the fact that datasets consisting of only dissimilarities can be used. It is important to note that we assume no additional bias node exists in the hidden layer because the objective function will be invariant with respect to this bias. We then perform a secondary linear mapping from the hidden layer to the target space, as given earlier in Eq. 2.6.

It is interesting to note that RBF-SNLM can specialize to Classical-SNLM by using the RBF

$$\phi_h(\mathbf{x}, \mathbf{c}_h, s_h) = \begin{cases} 1 & : \mathbf{x} = \mathbf{c}_h \\ 0 & : \text{ otherwise} \end{cases} \tag{3.2}$$

This makes the weights matrix effectively equivalent to the target configuration optimized in Classical-SNLM.

The parameters that can be adapted in the training algorithm are the weights $\mathbf{W}$, the spreads $\mathbf{s}$, and the centers matrix,

$$\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_h, \cdots, \mathbf{c}_H\}^T \tag{3.3}$$

where $\mathbf{c}_h$ is the center vector for hidden unit h. Of course, one can choose to initialize a parameter to a specific value and only adapt the remaining parameters.

3.2 Problem Formulation

The task of implementing a RBF neural network in order to perform SNLM requires the minimization of a suitable objective function. The Classical-SNLM procedure initializes a random configuration of points in the target space and adapts these points until the errors between the inter-point distances in the source space and the corresponding distances in the target space are minimized. This is accomplished by minimizing the *stress function* defined below:

$$E(\mathbf{D}) = \frac{1}{2} \sum_{i<j} u_{i,j} \left(d_{i,j} - \delta_{i,j}\right)^2 \tag{3.4}$$

where $d_{i,j} \in \mathbf{D}$ and $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the dissimilarity matrix containing all inter-point distances of the patterns in the target vector. $\delta_{i,j} \in \mathbb{R}$ is the dissimilarity between pattern i and pattern j of the training set in the original feature space. $u_{i,j}$ is the adjacency value for pattern i and pattern j, which is discussed in more detail in section 3.6.

Unless it is specified otherwise, the distance metric used in this document is the Euclidean distance

$$d_{i,j} = \|\mathbf{x}_i - \mathbf{x}_j\|_2 \tag{3.5}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are two patterns in the training set. We assume that the target patterns are produced by a non-linear basis model as defined in Eq. 2.6. The RBF used in all of our experiments with RBF-SNLM is the Gaussian RBF defined in Eq. 3.1.

As mentioned earlier, our proposed algorithm for implementing RBF-SNLM, shown in psuedocode in Alg. 1, uses numerical optimization to adapt the parameters of the model such that the stress function in Eq. 3.4 is minimized. The spreads and centers determine the interpolation matrix $\mathbf{\Phi}$. Multiplying $\mathbf{\Phi}^T$ by $\mathbf{W}$ yields the target vector, which in turn determines the target distance matrix $\mathbf{D}$. As such, the stress function is dependent on the input dataset and, indirectly, on the weights, spreads and centers parameters.

To train the network, we use off-line (batch) learning, where we compute the gradient of all parameters and the search directions before updating them for the current epoch. We adapt each parameter type (weights, spreads, and centers) separately, because different parameters may require very different step lengths. That is, only after the search directions and step lengths of all parameters are calculated, the parameters are updated. This order makes certain that each parameter adaptation is for the same epoch.

We have experimented with a number of optimization methods, all of them dependent on the gradient of the stress function with respect to each parameter. We have tried experimenting with the simple gradient descent, conjugate gradient descent (see subsection 3.3.1), and the limited memory BFGS quasi-Newton method. BFGS quasi-Newton (see subsection 3.3.2) approximates second-order information and can find a better search direction for ill-conditioned problems. This last technique is what we have been typically using to perform experiments with RBF-SNLM.

Once the search direction is known, RBF-SNLM performs a line search optimization to determine a near-optimal step length α by which to adapt the parameters. We have used in our algorithm a line search returning step lengths that satisfy the strong Wolfe conditions, but have also previously used less efficient algorithms such as

backtracking and constant step length. After adapting the parameters, we check the stopping condition to see if the training algorithm has converged. As a stopping condition we check the largest gradient values of the parameters to see if they are small enough for the algorithm to have converged.

input : Δ: A $\mathbb{R}^{N \times N}$ matrix of dissimilarities derived from the training set
output: $\mathbf{W}$: $\mathbb{R}^{H \times M}$ matrix storing the weight scalars
output: $\mathbf{s}$: $\mathbb{R}^{H \times 1}$ vector storing the positive spread scalars
output: $\mathbf{C}$: $\mathbb{R}^{H \times N}$ matrix storing the center vectors $\mathbb{R}^{N \times 1}$

// Initialize parameters;
$\mathbf{W} \leftarrow$ initializeWeights();
$\mathbf{s} \leftarrow$ initializeSpreads();
$\mathbf{C} \leftarrow$ initializeCenters();
// Update distance matrices;
$\mathbf{\Phi} \leftarrow$ `calculateInterpolationMatrix`$(\mathbf{W}, \mathbf{s}, \mathbf{C})$;
$\mathbf{Y} \leftarrow \mathbf{\Phi}^T \times \mathbf{W}$;
// Inter-point distances in target vector;
$\mathbf{D} \leftarrow$ `calculateDistance`$(\mathbf{Y})$;
// Calculate Residuals matrix;
$\mathbf{\Gamma} = \mathbf{D} - \mathbf{\Delta}$;
while *Stopping criterion is not met* **do**
    // Calculate optimization direction and step length for parameters;
    foreach *Parameter* $\boldsymbol{\theta}$ **do**
        $\nabla_\theta E \leftarrow$ `getGradient`$(\boldsymbol{\theta}, \mathbf{\Gamma}, \mathbf{D})$;
        $\mathbf{p} \leftarrow$ `getDirection`$(\theta, \nabla_\theta E)$;
        $\alpha \leftarrow$ `lineSearch`$(\boldsymbol{\theta}, \mathbf{p}, \mathbf{\Gamma}, \mathbf{D})$;
    end
    // Adapt parameters;
    foreach *Parameter* $\boldsymbol{\theta}$ **do**
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \mathbf{p}$;
    end
    // Update distance matrices;
    $\Phi \leftarrow$ `calculateInterpolationMatrix`$(\mathbf{W}, \mathbf{s}, \mathbf{C})$;
    $\mathbf{Y} \leftarrow \mathbf{\Phi}^T \times \mathbf{W}$;
    $\mathbf{D} \leftarrow$ `calculateDistance`$(\mathbf{Y})$;
    // Calculate Residuals matrix;
    $\mathbf{\Gamma} = \mathbf{D} - \mathbf{\Delta}$;
end
return $\mathbf{W}, \mathbf{s}, \mathbf{C}$

Algorithm 1: Training Algorithm for RBF-SNLM. Trains a model to be able to approximate a Sammon mapping.

There are auxiliary variables in the training algorithm: the interpolation matrix $\mathbf{\Phi}$, the target matrix $\mathbf{Y}$ and the distance matrix $\mathbf{D}$. Recall from Eq. 2.4 that $\mathbf{\Phi}$ is the matrix containing all values of the RBF functions evaluated for each training pattern. The $\mathbf{Y} \in \mathbb{R}^{N \times M}$ matrix contains all patterns in the target space.

$\mathbf{D} \in \mathbb{R}^{N \times N}$ is a matrix containing all inter-point distances of the patterns stored in the rows of $\mathbf{Y}$. These matrices are updated each epoch and passed to the *getGradient* function in the optimization section of Alg. 1.

3.3 Optimizing the Parameters

The parameters optimized in RBF-SNLM using the Gaussian RBF in Eq. 3.1 are the weights $\mathbf{W}$, spreads $\mathbf{s}$, and centers $\mathbf{C}$. The spreads and centers are parameters of the RBFs in the hidden layer. The weights are used in the activation functions of the output layer. Recall from Eq. 3.3 the center vector stores the centers for each hidden unit. When only the dissimilarities are available from the training set, adapting the centers is not possible, and the centers must be fixed to training patterns. In this case, the center parameter can be thought as a vector $\mathbf{c} \in \mathbb{R}^{H\times 1}$ containing the indices of the center patterns. The following subsections describe two methods for optimizing the model's parameters.

3.3.1 Conjugate gradient descent method

Batch learning methods-based online searches involve two steps. First, the search direction vector is calculated, typically by utilizing the gradient of the stress function with respect to a parameter $\boldsymbol{\theta}$. Next, an appropriate step length along the determined search direction is identified so that the stress function is (most of the time, approximately) minimized. After the search directions and step lengths have been determined for every parameter, the parameters are adapted.

The *gradient descent method* determines the search direction to be the negative gradient. If we define $\mathbf{p}_k$ to be the search direction for epoch k then the gradient descent method uses:

$$\mathbf{p}_k = -\nabla_\theta E_k \tag{3.6}$$

Then the parameter is adapted in the direction $\mathbf{p}_{k+1}$ by a near-optimal step length α.

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \alpha \mathbf{p}_k \tag{3.7}$$

The determination of $\mathbf{p}_k$ only depends on the current parameter's value and the derivative of the objective function at the current configuration, which does not always provide accurate information about the location of the local minimum. In particular, gradient descent does not factor in any curvature information, when determining a promising search direction.

As a more efficient alternative to gradient descent, we apply the *conjugate gradient descent (CGD) method*[15]. CGD computes a series of search direction vectors $\mathbf{p}_1, \cdots, \mathbf{p}_M$ that are conjugate to each other. If, in the vicinity of the current solution, the stress function can be well approximated by a convex quadric function, then the last adaptation should place the parameter $\boldsymbol{\theta}$ in a configuration such that the stress function E is a minimum with respect to $\boldsymbol{\theta}$. The accuracy of CGD depends on how near-convex quadric the stress function is and how exact the line search is. In an effort to make the method more robust to deviations from the convex quadric approximation, the search direction vector is reset to the negative gradient vector every M epochs. The formulas for conjugate gradient follow:

$$\mathbf{p}_k = \begin{cases} -\nabla_\theta E_k & : \text{ k mod } M = 0 \\ -\nabla_\theta E_k + \beta_k * \mathbf{p}_{k-1} & : \text{ otherwise} \end{cases}$$

$$\beta_k = \frac{\nabla_\theta E_k^T \nabla_\theta E_k}{\nabla_\theta E_{k-1}^T \nabla_\theta E_{k-1}} \tag{3.8}$$

The β value is a constant chosen such that it makes the direction vector $\mathbf{p}_k$ conjugate to the previous direction vector $\mathbf{p}_{k-1}$.

3.3.2 Quasi-Newton methods

In addition to the conjugate gradient descent approach for finding the direction vector, we have also used an algorithm belonging to the family of *quasi-Newton (QN) methods*. QN methods are iterative optimization methods that attempt to behave like Newton's Method near the minimum but do not have the disadvantage of a $O(N^3)$ (where N is the number of parameters being optimized) complexity, neither in time nor in space. Instead of calculating the Hessian like Newton's method, a QN optimization algorithm uses the changes in the gradient to estimate the local curvature and produce an approximate Hessian at that configuration. If a near-optimal step length in the search direction is found, such as one satisfying both Wolfe conditions, then training should approach super-linear convergence and as a result, it could be much faster than simple conjugate gradient descent[15].

The particular QN variant we implemented for RBF-SNLM was the BFGS QN method, named after its four creators Broyden[16], Fletcher[17], Goldfarb[18], and Shanno[19]. At each iteration k, we write the quadric minimizer of Eq. 3.9 where $\mathbf{B}_k \in \mathbb{R}^{N\times N}$ is a symmetric matrix updated at each iteration of BFGS. The direction vector is then calculated by Eq. 3.10. The fundamental difference between BFGS and Newton's method is that instead of calculating the exact Hessian matrix every iteration, BFGS calculates $\mathbf{B}_k$, which is an iterative approximation to the Hessian matrix.

$$m_k(\mathbf{p}_k) = E_k + \nabla E_k^T \mathbf{p}_k + \frac{1}{2}\mathbf{p}_k^T \mathbf{B}_k \mathbf{p}_k \tag{3.9}$$

$$\mathbf{p_k} = -\mathbf{B_k^{-1}}\nabla E_k \tag{3.10}$$

In order to update $\mathbf{B_k}$ to $\mathbf{B}_{k+1}$, we impose the conditions that the gradient of the error function E at the k^{th} and $(k+1)^{th}$ iteration must match the gradient of the minimizer function m at the $(k+1)^{th}$ iteration. The second condition is automatically satisfied for $\alpha_k = 0$ and the first one results in Eq. 3.11. If we define $\Delta\boldsymbol{\theta}_k \hat{=} \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k$ and $\Delta\mathbf{g}_k \hat{=} \nabla E_{k+1} - \nabla E_k$, then we obtain the secant equation (Eq. 3.12) and we may maintain the curvature condition (Eq. 3.13) since $\mathbf{B_k}$ is a positive symmetric matrix.

$$\nabla m_{k+1}(-\alpha_k \mathbf{p}_k) = \nabla \mathbf{E}_{k+1} - \alpha_k \mathbf{B}_{k+1} p_k = \nabla \mathbf{E}_k \tag{3.11}$$

$$\mathbf{B}_{k+1}\Delta\boldsymbol{\theta}_k = \Delta\mathbf{g}_k \tag{3.12}$$

$$\boldsymbol{\Delta\theta}_k^T \Delta\mathbf{g}_k > 0 \tag{3.13}$$

Since there are many such $\mathbf{B}$, we require that we find $\min_B \|\mathbf{B} - \mathbf{B}_k\|$ such that the secant equation is satisfied and $\mathbf{B}_k$ remains a symmetric matrix. We then have an inverse matrix $\mathbf{H_k} \equiv \mathbf{B_k^{-1}}$ and an iteration of this matrix is given by Eq. 3.14

$$\mathbf{H_{k+1}} = (\mathbf{I} - \rho_k \boldsymbol{\Delta\theta_k} \mathbf{\Delta g_k^T})\mathbf{H_k}(\mathbf{I} - \rho_k \boldsymbol{\Delta\theta_k} \mathbf{\Delta g_k^T}) + \rho_k \boldsymbol{\Delta\theta_k} \boldsymbol{\Delta\theta_k^T} \tag{3.14}$$

for $\rho_k \equiv \frac{1}{\mathbf{\Delta g_k^T \Delta\theta_k}}$.

Fig. 2 illustrates the time, space and convergence rate trade-off among the two methods we used and the regular gradient descent method. The plot illustrates that QN methods are able to speed up convergence at an increased spatial (storage) and computational cost. Whenever the fitted RBF-SNLM model has too many parameters, employing QN methods might not be feasible; under these circumstances CGD methods are always preferred over plain GD.

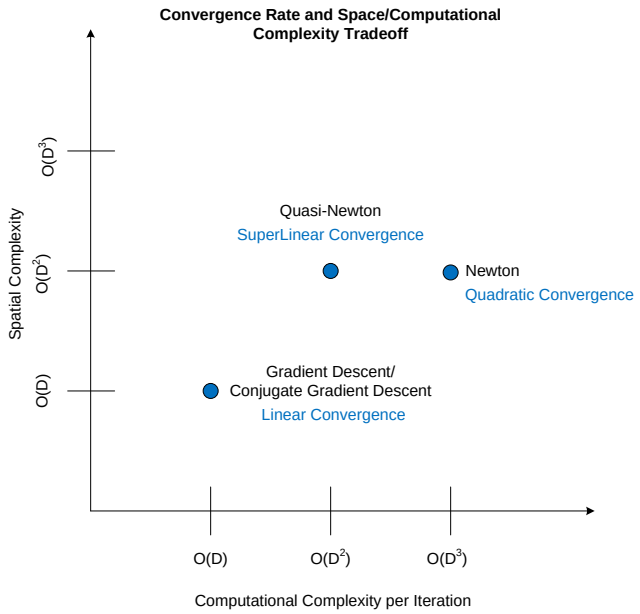

Figure 2. Time, space complexity and convergence rate trade-off for conjugate gradient descent, gradient descent, quasi-Newton method, and Newton's method. Fast convergence is achieved at the cost of higher computational and spatial complexity of the algorithms involved.

3.3.3 Line search

In training the model to construct approximate isometries, RBF-SNLM uses a line search to find a near-optimal step length in the direction of $\mathbf{p}_k$. In essence, the line search is a one-dimensional optimization problem parameterized by the step length along the current search direction vector, so that the stress function is being minimized. Once the search direction is calculated by an optimization method, the step length α needs to be determined. Too small a step length and the optimization algorithm would take too much time as progress would be very slow; too large a step length would overshoot the minimum.

It would be ideal to be able to find the optimal step length, α_o, such that $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_o \mathbf{p}_k$ would be as close as possible to the minimum within the limitations of precision. However, in practice, the so called "inexact" line searches (ones that find near-optimal step lengths) are almost always preferred over their "exact" counterparts. The reason behind this is that the precision in determining the minimizer by performing exact line searches is usually outweighed by the accruing, costly computational effort involved: overall, inexact line searches may accomplish the same progress towards a local minimum as exact methods, but with significantly less computational burden.

The line search RBF-SNLM uses returns a step length that satisfies the strong Wolfe conditions. A step length α that satisfies the Wolfe conditions meets the criterion for near-optimality. In this context, the value of the stress function evaluated at the new configuration (the configuration after the adaptation of $\boldsymbol{\theta}$ with step length α) has sufficiently decreased.

The Wolfe conditions test the stress function E at the trial step length α with some constants $0 < c_1 < c_2 < 1$ for both sufficient decrease and for appropriate curvature. Let

$$\varepsilon(\alpha) \hat{=} E(\boldsymbol{\theta}_k + \alpha \mathbf{p}_k^T) \tag{3.15}$$

be the stress function value at point $\boldsymbol{\theta}_k$, when a step of size α is taken along direction $\mathbf{p}_k$. The first Wolfe condition deals with the amount of decrease in E and states

$$\varepsilon(\alpha) \leq \varepsilon(0) + c_1 \alpha \varepsilon'(0) \tag{3.16}$$

Here c_1 is a user-defined parameter. Typically this value is small in its range, specifically $c_1 < 0.1$. This condition stipulates that the function E must exhibit sufficient decrease after the next step. Specifically, the function at α_{k+1} must be below the line formed at the intercept $\varepsilon(0)$ with the slope equal to the slope at $\varepsilon(0)$. When E is reduced sufficiently, progress will be made toward the minimum.

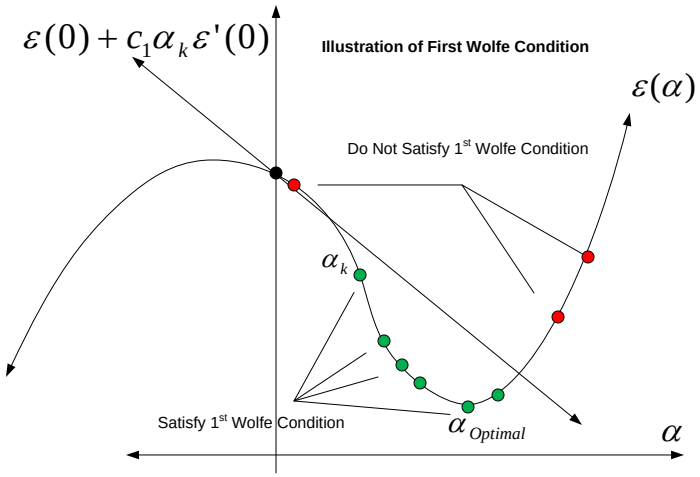

Figure 3. 1^{st} Wolfe Condition: Step lengths lie under the sufficient decrease line to pass the first condition.

This first condition does not yield a good step length in all cases; it will do poorly in cases where the function is very steep around the minimum so that a large change in E at the next step may yield no significant progress toward the minimum.

The second Wolfe condition deals with the curvature in E and states

$$|\varepsilon'(\alpha)| \leq c_2\varepsilon'(0) \tag{3.17}$$

c_2 is a user-defined parameter. Typically, it is desirable to set its value large in its range, specifically $c_2 > 0.9$. The second condition stipulates that the chosen step length α must yield a next step that has sufficient reduction in the slope of E. When the slope is reduced sufficiently, a local minimum of E in the direction $\mathbf{p}_k$ is reached.

Combining these two conditions will yield a good step length in most cases, giving both sufficient decrease in the objective function E and sufficient decrease in its slope as well.

The algorithm that RBF-SNLM uses (shown in Alg. 2) to find a step length that satisfies both strong Wolfe conditions is based on an algorithm from Nocedal[15]. The algorithm is somewhat complex and requires explanation. It starts with a parameter $\boldsymbol{\theta}_k$ on the function E that is undergoing optimization. It must choose a step length that satisfies both Wolfe conditions. Then the next step, when taken, will yield a value that has sufficient decrease in the cost function E and is sufficiently closer to the minimum. The Wolfe-based line search chooses a trial step length in the interval from α_{prev} and α_{max}. Based on this trial step length and the previous trial step length, it goes through three cases that will yield a sufficient final step length.

The first case checks to see if the current trial step length α_{cur} can be an upper bound for the final step length. The line search does this by checking whether α_{cur} fails to satisfy the first Wolfe condition for sufficient decrease or whether this step yields a value from E that is greater than the previously found trial step length α_{prev}. In the first case a minimum must exist between $\alpha = 0$ and $\alpha = \alpha_{cur}$. In the second case, a minimum must exist between $\alpha = \alpha_{prev}$ and $\alpha = \alpha_{cur}$. The *zoom* function presented in Alg. 3 and described later in subsection 3.3.4, attempts to find a step length between these two bounds so that when the step is taken, will yield a point $\boldsymbol{\theta}_{k+1}$ that is sufficiently close to the minimum.

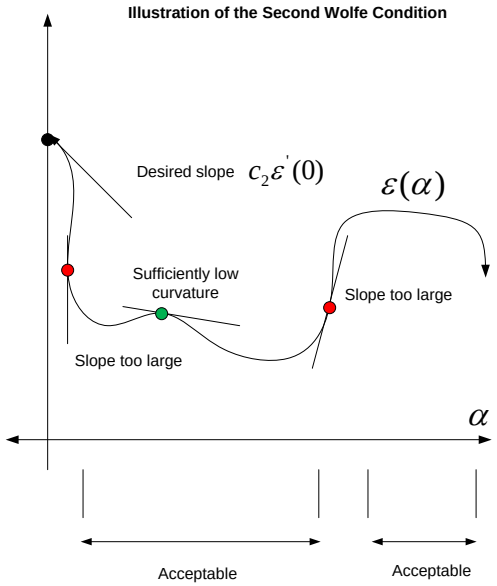

Figure 4. 2^{nd} Wolfe Condition: $\varepsilon'(\alpha)$ must be sufficiently less at α = trial step than at $\alpha = 0$.

```
1  function LineSearch
   input  : θ: current value of the parameter being adapted
   output: α: step length that satisfies the strong Wolfe conditions
2  α_prev ← 0;
3  α_cur ← random number between 0 and α_max;
4  onceThroughFlag = false;
5  while α_cur ≤ α_max do
6      if ε(α_cur) > ε(0) + c1 α_cur ε'(0) or (onceThroughFlag == true and ε(α_cur) ≥ ε(α_prev)) then
7          α ← zoom(α_prev, α_cur);
8          return;
9      end
10     if |ε'(α_cur)| ≤ −c2 ε'(0) then
11         α ← α_cur;
12         return;
13     end
14     if ε'(α_cur) ≥ 0 then
15         α ← zoom(α_prev, α_cur);
16         return;
17     end
18     α_prev ← α_cur;
19     α_cur ← select a step length from the interval (α_cur, α_max);
20     onceThroughFlag ← true;
21 end
```

Algorithm 2: Wolfe-Based Line Search finds a step length that satisfies both strong Wolfe conditions for sufficient decrease in the stress function and sufficient decrease in the directional derivative.

If the first case fails and the trial step length α_{cur} satisfies the first Wolfe condition for sufficient decrease, the step length is checked against the second Wolfe condition to see if $\varepsilon(\alpha_{cur})$ is sufficiently small (flat). If α_{cur} satisfies both of these conditions, the line search has found an appropriate step length and returns.

If α_{cur} yields a sufficient decrease but fails to yield a value that is sufficiently closer to the minimum, it may have overstepped the minimum. If this is true, the derivative at α_{cur} will be positive. However, since α_{cur} sufficiently decreases E, α_{cur} can be used as a tighter lower bound. The interval between α_{cur} and the previous trial step length α_{prev} contains an ideal step length and *zoom* is used to find it.

```
function zoom
input  : Values α_lo and α_hi
output: α between α_lo and α_hi that aims to minimize ε(α)
// Set threshold value;
while α_lo ≠ α_hi do
    α_j ← Cubicinterpolate(α_lo, α_hi) ;
    Evaluate(ε(α_j)) ;
    // Checking to see whether Wolfe Condition 1 is violated OR
    // whether a new upper bound for ε can be determined
    if (ε(α_j) > ε(0) + c_1 α_j ε'(0)) or (ε(α_j) ≥ ε(α_lo)) then
        α_hi ← α_j;
    end
     else
        Evaluate(ε'(α_j)) ;
        // If both Wolfe Conditions are satisfied, then α_j is near-optimal;
        if |ε'(α_j)| ≤ −c_2 ε'(0) then
            α ← α_j ;
        end
        if ε'(α_j)(α_hi − α_lo) ≥ 0 then
            α_hi ← α_lo;
        end
        α_lo ← α_j;
    end
end
```

Algorithm 3: *Zoom* decreases the boundary region found by the Wolfe line search around the optimal step length. It attempts to find a near-optimal step length with the cubic interpolation from Alg. 4.

```
input  : α_lo: Step length that is the lower boundary for ε(α)
         α_hi: Step length that is the upper boundary for ε(α)
output: α: Step length that attempts to minimize ε(α)
if α_hi < α_lo then
    swap α_lo and α_hi;
end
d_1 ← ε'(α_lo) + ε'(α_hi) + 3(ε(α_lo) − ε(α_hi))/(α_lo − α_hi);
d_2 ← sqrt(d_1^2 − ε'(α_lo)ε'(α_hi));
α ← α_hi − (α_hi − α_lo)(ε'(α_hi) + d_2 − d_1)/(ε'(α_hi) − ε'(α_lo) + 2d_2);
```

Algorithm 4: Cubic interpolation to find the minimum of a cubic-like function given its boundaries and first-order information about the function at given points

3.3.4 Explanation of the *zoom* procedure

Here, we explain the *zoom* subroutine pseudocode in Alg. 3. The strategy is that zoom repeatedly performs the cubic interpolation algorithm of Alg. 4 on the function between two points to find a minimizer that satisfies the two Wolfe conditions. We begin first with the outermost If-Block. In Fig. 5 below, we are given $\varepsilon(\alpha)$, where the two reference points, α_{lo} and α_{hi} are marked in black. Interpolation, of any kind, could theoretically yield the three step lengths marked in green and red. Only the green step length, α_{j1}, falls within the range stipulated by the first Wolfe condition. Outside of this range, step lengths will fail the first Wolfe condition because they do not yield sufficient decrease in the stress function. We encounter the following three cases:

1. The new α found satisfies the sufficient decrease Wolfe condition AND $\varepsilon(\alpha) < \varepsilon(\alpha_{lo})$, then nothing is done.
2. The new α found violates the sufficient decrease Wolfe condition. In this case, α is assigned to be the new α_{hi} since this point sets a new upper boundary on α_{hi}.
3. The new α, regardless of whether it satisfies the sufficient decrease Wolfe condition or not, has a function value $\varepsilon(\alpha) \geq \varepsilon(\alpha_{lo})$. α is again assigned to be the new α_{hi} since this point also sets a new upper boundary on α_{hi} . See point α_{j3}.

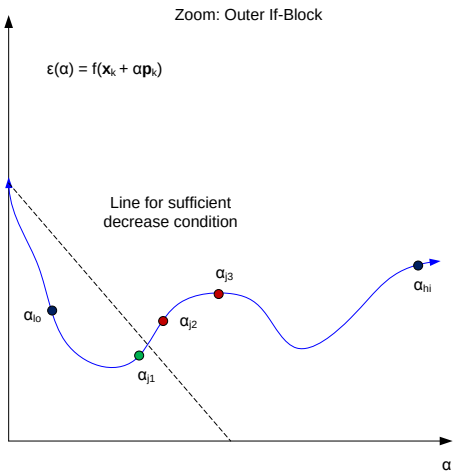

Figure 5. Zoom - Finding a suitable α to satisfy the first Wolfe Condition.

Next we proceed to examine the Else-Block. Here, one of two cases could happen. Note that the setup in Fig. 6 is essentially the same as in the previous figure except that the function $\varepsilon(\alpha)$ itself is different and $\varepsilon'(0)$ is drawn to illustrate the concept. Note that at this stage, the sufficient decrease Wolfe condition has been satisfied. Here, one of two cases could happen:

1. The strong second Wolfe curvature condition is satisfied. Then the search for the optimal α terminates and α is recognized as the optimal step length.
2. The strong second Wolfe curvature is not satisfied due to too sharp of a gradient. Then if the slope of $\varepsilon'(\alpha_j)$ is nonnegative and $\alpha_{hi} > \alpha_{lo}$ or if the slope of $\varepsilon'(\alpha_j)$ is non-positive and $\alpha_{lo} > \alpha_{hi}$, then α_{hi} and α_{lo} are swapped since this is an indication that we are searching in the wrong direction (i.e. we have already passed the minimum going from smaller to larger values in the first case or larger to smaller values in the second case). So the search continues and a new α_j is interpolated. Note that the key detail to realize here is that α_{lo} is not necessarily less than α_{hi}, but that the stress function has a lower value at α_{lo} than at α_{hi}.

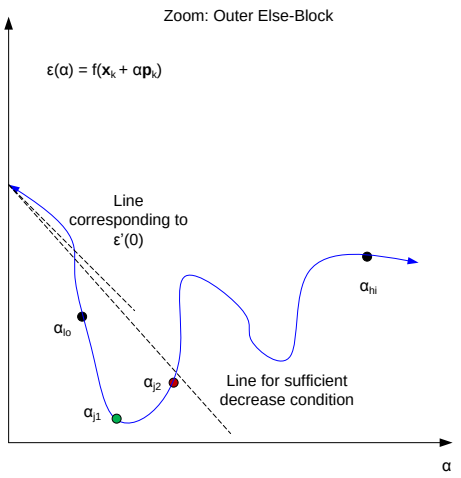

Figure 6. Zoom - After the sufficient decrease Wolfe condition is satisfied, then the outer Else-Block will check for the curvature Wolfe condition.

3.3.5 Gradient Equations

All of our optimization methods require the calculation of the stress function with respect to the optimization parameters. One can generalize the gradient of any parameter to Eq. 3.18.

$$\frac{\partial E}{\partial \theta} = \frac{1}{2} \sum_{i<j} u_{i,j} \left(1 - \frac{\delta_{i,j}}{d_{i,j}}\right) \frac{\partial d_{i,j}^2}{\partial \theta} \tag{3.18}$$

In our particular case, we are optimizing $\mathbf{W}$, $\mathbf{s}$ and $\mathbf{C}$, whose gradient equations are given below. The gradient for the weights:

$$\frac{\partial E}{\partial \mathbf{w}_k} = \left[\sum_{i<j} u_{i,j} \left(1 - \frac{\delta_{i,j}}{d_{i,j}}\right) \Phi(\mathbf{x}_i, \mathbf{x}_j)\right] \mathbf{w}_k \tag{3.19}$$

The term for the spreads:

$$\frac{\partial d_{i,j}^2}{\partial s_h} = 2 \left[\frac{\partial \phi_h(\mathbf{x}_i)}{\partial s} - \frac{\partial \phi_h(\mathbf{x}_j)}{\partial s}\right] \sum_{m=1}^{M} \left(w_m w_m^T\right) \left[\phi_h(\mathbf{x}_i) - \phi_h(\mathbf{x}_j)\right] \tag{3.20}$$

The term for the centers:

$$\frac{\partial d_{i,j}^2}{\partial \mathbf{c_h}} = \left[\frac{2}{s_h}(\phi_h(\mathbf{x}_i)(\mathbf{x}_i - \mathbf{c}_h) - \phi_h(\mathbf{x}_j)(\mathbf{x}_j - \mathbf{c}_h))\right] \left[\sum_{i<j}^{M} w_{h,m} \mathbf{w}_m^T\right] \left[\phi_h(\mathbf{x}_i) - \phi_h(\mathbf{x}_j)\right] \tag{3.21}$$

3.4 Advantages of RBF-SNLM over Classical SNLM

In contrast to Classical-SNLM, the RBF-SNLM technique allows for the interpolation or extrapolation of data points not in the original training pattern dataset because once $\mathbf{W}$, $\mathbf{s}$, and $\mathbf{C}$ are known, we can easily find the location of the points in the target database. The interpolation or extrapolation can be improved by adjusting the number of hidden node, as described in section 4, to make the model generalize better.

Furthermore, adapting all these aforementioned parameters allows for accurate tuning of the input's images. As mentioned earlier, another advantage of RBF-SNLM is that we need only know the dissimilarity between two data points or the data point and center and not the actual patterns themselves. That is, with the centers fixed to the training points in the training dataset, nowhere does the training algorithm require knowledge of what the actual points are themselves. Then, we may work purely with dissimilarities and explore non-metric datasets whose dissimilarities may hold much more significance than the actual locations of the points.

3.5 Multiple Minima

In SNLM, the solutions to the reduction problem are not unique because they are rotation- and translation-invariant. In RBF-SNLM, this property is still true. It is straightforward to visually inspect that given any two target configurations of points C_1 and C_2, where C_2 is a translation or rotation of C_1, the value of the objective function evaluated at C_1 will be identical to that of C_2. This is because the objective function depends on the differences of the distances between points in the target space and the source space, and not on the points themselves.

Of course, it may also be that the solution produced by the training algorithm is only locally-optimal. In numerical optimization, the global solution is usually too difficult to determine. Since many parameters are being optimized in RBF-SNLM, it may be that there are multiple local minima corresponding to output configurations that are not rotations or translations of each other. In this case, the objective function evaluated for these configurations would most likely be different. This particular issue could be potentially addressed by restarting the training using different initial parameter values so that different locally-optimal configurations are produced.

3.6 Adjacency Matrices

It is sometimes advantageous to ignore certain pair-wise distances between training patterns if these distances are deemed irrelevant or not as important to a particular visualization task. If desired by the user of RBF-SNLM, certain distances in the dataset can be totally ignored or weighted so that their errors are less contributing to the stress function. RBF-SNLM can handle such cases through use of an *adjacency matrix*, $\mathbf{U}$, which weights appropriately the residuals in the stress function. Ideally, given some previous knowledge about the dataset, we would like to manipulate the entries of this matrix to provide meaningful visualization results in the space of reduced dimensions. Furthermore, sparse $\mathbf{U}$ matrices may significantly reduce the computational complexity of RBF-SNLM because fewer terms in Eq. 3.18 are zero. Unfortunately, the determination of $\mathbf{U}$ is very dependent on the dataset being examined and the user's goals. Moreover, the user must know some general structure of the input data in order to take advantage of the adjacency matrix. This subsection aims to present a few general adjacency matrices that can be used on a wide variety of datasets.

The standard adjacency matrix used is the all-ones matrix. For a dataset consisting of N patterns, this matrix is simply of size $N \times N$ with each entry equal to 1 so that all dissimilarities between the patterns are taken into account with equal weight. This is the default approach to use when no previous information is known about the dataset. Note, however, that using an all-ones adjacency matrix forces all stress function terms to be taken into account, which could be computationally very expensive.

For visualization purposes, we explore the use of several graph matrices explained as follows. An alternate consideration to the all-ones adjacency matrix is the ϵ matrix, which preserves dissimilarities between points i and $j \iff dist(i,j) < \epsilon$ for ϵ being some small, positive constant. In other words, the adjacency matrix entry associated with i and j is nonzero if pattern j lies in the ϵ-neighborhood (with respect to $dist$ and $dist$ being some kind of dissimilarity metric) of pattern i. One particular usage of this matrix that has been demonstrated in the literature is its ability to unfold the classical *Swiss roll* dataset[20]. The adjacency matrix is depicted visually in the Fig. 7. Typical values for epsilon are 0.3 and 0.8 for normalized data[15]. Subsection 3.8 deals further with normalization of datasets.

A similar matrix to the ϵ matrix is the τ matrix. The conditions to be satisfied for a graph connection are slightly more complex. Specifically, given two distances $d_{i,min} = \min_j \delta_{i,j}$ and $d_{min,j} = \min_i \delta_{i,j}$ for $d_{k,min}$ denoting the closest point to point k in the entire set, a graph connection is only formed if special similarity and neighborhood conditions are satisfied. In specific, the similarity connection requires that $d_i \leq \tau d_j$ and $d_j \leq \tau d_i$ for some positive tolerance τ so that these nearest-neighbor distances are of the same order of magnitude. That is, $dist(i,j) \leq \tau d_i$ or, likewise, $dist(i,j) \leq \tau d_j$[15].

Another variation, the K matrix, does not utilize the absolute pairwise dissimilarities between data patterns but instead, relies on the rank order dissimilarities between each pair of patterns to compute its graph connections. That is, each pattern simply forms a connection with its K- nearest neighbors.[15] This is rather simple to construct since the dissimilarity between each pair of patterns is known from the onset of the minimization. However, sorting of the dissimilarities must still be performed to realize their relative sizes.

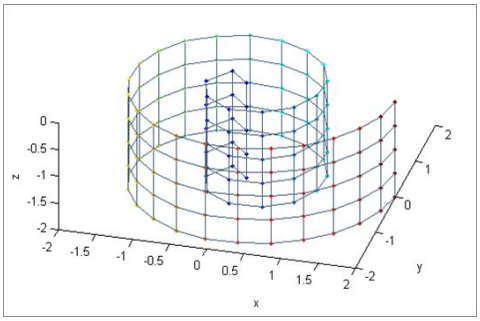

Figure 7. Epsilon matrix applied to the *Swiss roll* dataset. Lines indicate adjacency connections.

A more complicated variation of the K - rule is the data rule, used to form the data adjacency matrix. Generally, a better extraction of the overall shape of the data set being visualized can be achieved with a rule such as this, which incorporates the ellipse and circle conditions. However, in these cases, the center prototypes must be known. Then, for points x_i and x_j and centers c_r and c_s, the ellipse condition imposes the restriction $dist(i, c_r) + dist(j, c_s) < C_1 dist(c_r, c_s)$ so the points must lie inside an ellipse with foci defined by the centers c_r and c_s. Additionally, the circle condition requires that $dist(i, c_r) < C_2 dist(i, c_s)$ and similarly, $dist(i, c_s) < C_2 dist(i, c_r)$. The constants $C_1 \equiv \sqrt{s^2 + 1}$ and $C_2 \equiv \frac{1+s}{1-s}$ where s is a parameter taking values in [0.2, 0.6][15].

3.7 Initialization of Parameters

Initialization of the parameters plays a substantial role in the final mapping of the dataset. Since we are interested primarily in the visualization, we normalize the patterns in the original database, if available, so that each feature is zero-centered and has a standard deviation of one. Furthermore, since the RBF function depends on the dissimilarities, we divide each entry of the original database by the maximum entry in the calculated dissimilarity matrix. The entries of the **W** matrix are then randomly generated numbers in the interval (0, 1). Likewise, the entries of the **s** parameter, if they are all distinct (each RBF features its own spread parameter), are randomly generated numbers in the interval (0,1). Note that no value in **s** can be 0 since it is in the denominator of the Gaussian RBF. If a common spread parameter s is used for all RBFs, only one random number in the interval (0,1) is chosen. The centers **C** may be initialized randomly, set to the training patterns, or initialized with the *k-medoids* clustering algorithm[21].

K-medoids is a clustering algorithm where the set of cluster centers is a subset of the input patterns[21]. It takes as input a set of patterns and a number of clusters k $\leq N$, where N is the number of patterns in the training set. It returns a clustering with k clusters, each consisting of at least one point. Each cluster has one and only one representative point, called the medoid of that cluster. Medoids $m_1..m_k$ must be distinct patterns within the input dataset. In the case of RBF-SNLM, where we must be able to perform reduction on datasets consisting only of distances, a modification to the k-medoids algorithm must be made. This modification is that instead of passing the patterns themselves, we pass to k-medoids the distance matrix $\mathbf{\Delta}$ containing all interpoint distances the training set. The algorithm returns the indices of the medoids.

3.8 Normalization

One particular use of MDS is the generation of an isometric mapping of patterns in the high dimensionality source space to a low dimensionality target space. However, for many of our datasets, in order to easily generate visually meaningful results, we perform some type of normalization. For instance, if we are interested only in the relative dissimilarities of the training patterns, a multiplication of the entire dataset by a scalar will not detract from the mapping.

Furthermore, in terms of working with non-metric data, we often choose to normalize the training patterns by making each feature zero-centered and having a standard deviation of one via an affine linear transformation. Note that in this case, each dimension is scaled individually and relative dissimilarities between dimensions are not preserved. There are several advantages to this. First, this allows for a uniform way to initialize the **W**,

s, and **C** parameters independently of the dataset being used. Similarly, this normalization also allows for the initial step lengths chosen for the Wolfe-based line search to be standardized. Furthermore, the derivative of the Gaussian kernel function becomes relatively saturated when it is too distant from the zero center. As a result, the derivatives evaluated at those points are small and the gradients calculated for the parameters are likewise small. As a result, the parameters would converge rather slowly. Normalizing the dataset aims at avoiding this problem. We use such a normalization technique when we find it fitting to extend the use of MDS from being simply an application on a metric network to one that is used on a topological network as well, as in the case of the *Teapots* dataset[22,23] (see Section 4.5).

3.9 Training modes

The training algorithm for RBF-SNLM is abstract enough to allow for different implementations that have varying capabilities depending on the parameters being optimized and the type of inputs. In this subsection, we describe the different modes of operation for the training algorithm.

The standard mode of training is to use a training set consisting of the patterns and adapt all of the parameters in the hidden layer and the weights of the output layer as well. For a Gaussian RBF, one would adapt the spreads, centers, and weights. The weights are initialized to arbitrary values within the range of the input dataset. The spreads are initialized to arbitrary positive real numbers whose range also depends on the input dataset, and the centers are initialized in a manner that chooses representative locations, again with respect to the given training set.

These locations can be chosen in a number of ways. Randomly choosing and clustering are useful methods. K-means or k-medoids can be used to find $k = H$ representative patterns. If $H = N$, then k-medoids reduces to simply choosing all training patterns to be the centers. This particular mode can be used to find a better approximate isometry for some training data, but may not be as accurate in interpolation due to overfitting. Also, the execution time of this type of RBF-SNLM will take the longest amount of time out of all modes.

A variation on the previous mode is when all parameters except the centers are adapted. The parameters are initialized like above. However, no optimization algorithm or line search is performed on the centers parameter. While this method may produce worse approximate isometries, the interpolation would be better due to the inability to overfit as much as when using the previous mode.

One of our main objectives is to be able to train a model with only the pair-wise dissimilarities derived from the training set while retaining the ability to interpolate new samples. This mode would adapt weights and spreads. Instead of using a collection of center vectors, this mode only requires the specification of a vector of training pattern indices that will serve as cluster centers/representatives. The indices are chosen by the same means as the above method. The centers are kept constant throughout the training process. It is future work to learn how to adapt them as well. This mode can reduce dissimilarity datasets and can map new dissimilarities as well.

4 RESULTS AND DISCUSSION

The basic use of RBF-SNLM is to map high dimensional data to lower dimensional 2D or 3D for visualization purposes. This can be done either by learning the mapping of the samples or an interpolation after the mapping is learned. In this particular section, we showcase the obtained results using RBF-SNLM on five datasets: *Square*, *Congressional Voting*, the *Federalist Papers*, the *Swiss roll*, and *Teapots*.

4.1 *Square Dataset*

As a check for the implementation of RBF-SNLM and as an exploration of the basic properties of the mapping, we begin with some preliminary results on a highly predictable example.

One of the key deviations between RBF-SNLM and Classical - SNLM is the introduction of adjustable centers for the RBFs. As a test case for the illustration of this particular concept, we introduce the *Square dataset*, as depicted in Fig. 8. This particular dataset is a simple array of 36 points in 2D sampled equidistantly over [2,11] in both the X and Y dimensions with each of square's four sides consisting of 10 data patterns (each of the four corners is shared by two sides). We use RBF-SNLM to remap this particular dataset from 2D to 2D. Since this

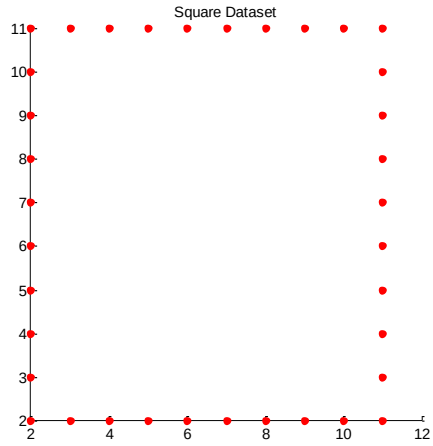

Figure 8. The original square dataset of 36 evenly spaced training patterns.

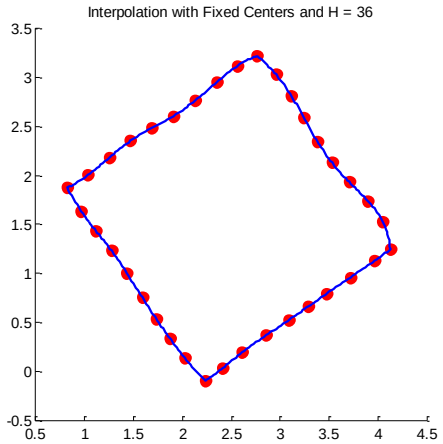

Figure 9. RBF-SNLM of the square dataset with fixed centers and $H = N = 36$. Red points depict mapping of the original dataset. Blue lines depict the corresponding interpolation of points along the square. *Specifics*: $\mathbf{W}$ and $\mathbf{s}$ parameters were randomly initialized. Each RBF featured its own spread parameter. Fixed centers were initialized via k-medoids. 150 iterations were run via quasi-Newton Method.

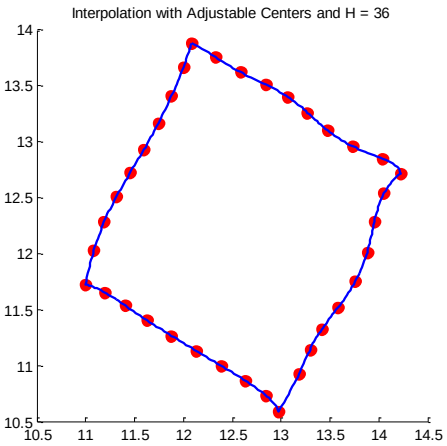

Figure 10. RBF-SNLM of the square dataset with adjustable centers and $H = N = 36$. *Specifics*: $\mathbf{W}$ and $\mathbf{s}$ parameters were randomly initialized. Each RBF featured its own spread parameter. Adjustable centers were initialized via k-medoids. 150 iterations were run via quasi-Newton Method.

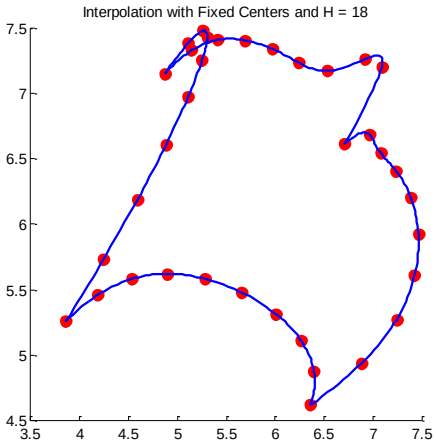

Figure 11. RBF-SNLM of the square dataset with fixed centers and $H = \frac{N}{2} = 18$. *Specifics*: $\mathbf{W}$ and $\mathbf{s}$ parameters were randomly initialized. Each RBF had its own spread parameter. Hidden dimension = 18. Fixed centers were initialized via k-medoids. 300 iterations were run via quasi-Newton Method.

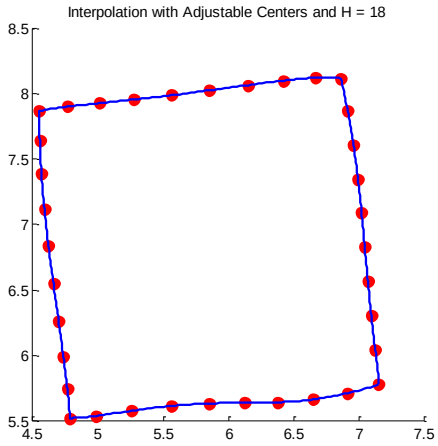

Figure 12. RBF-SNLM of the square dataset with adjustable centers and $H = \frac{N}{2} = 18$. *Specifics*: $\mathbf{W}$ and $\mathbf{s}$ parameters were randomly initialized. Each RBF had its own spread parameter. Adjustable centers were initialized via k-medoids. 150 iterations were run via quasi-Newton Method.

is a direct same-dimension mapping with no dimensionality reduction, it should be rather simple and distance preservation should be exact.

As a parallel between RBF-SNLM and Classical-SNLM, we set the number of hidden nodes in the RBF layer to be equal to the number of training patterns. In this particular case, the centers in the RBF layer can simply equal the training patterns in the original feature space. Then, with fixed centers, if the spreads, $\mathbf{s}$, approach $\mathbf{0}$, each RBF function responds with a 1 if the input pattern matches the center of the RBF (one of the training patterns). Otherwise it responds with a 0. In other words, the RBF becomes the *hit-or-miss* activation function as defined in Eq. 3.2. Then, it follows that $\mathbf{W}$ is analogous to the patterns in the target space obtained via Classical-SNLM. Note, however, that interpolation of points not in the original database is slightly better when the centers are adjustable so that over-training is reduced whenever the dimensionality of the hidden layer is less than the cardinality $|N|$ of the training set. Fig. 9 demonstrates the results of a RBF-SNLM mapping using fixed centers for the number of hidden nodes equal to the number of training patterns. The red dots represent the mapping of the points from the training set. The blue lines in between depict the interpolated points, which are line-connected for visualization purposes. Fig. 10 illustrates the result with adjustable centers. Note that in both cases, the points in the original feature space are mapped rather well, but the quality of generalization and the interpolation of novel points is not as good. Furthermore, there is no marked difference in the mapping and interpolation of points between the case where the centers are adjustable and the case where they are fixed. This is expected since using $H = N$ nodes in the hidden layer does not entail a loss in a degree of freedom for distance preservation as this is a 2D-to-2D mapping and inter-point distances can be perfectly preserved.

There is, however, a trade-off between the accuracy of the mapping and the quality of the interpolation. Having the number of hidden nodes in the RBF layer equal to the number of total training pattern provides a relatively accurate mapping but detracts from the results of interpolation, perhaps due to over-training. On the other hand, having a reduced number of hidden nodes in the RBF layer detracts from the mapping of the training set but improves on the interpolation quality of points not included as part of the original training patterns. Again, we illustrate this with a simple example of the square dataset. Here, the adjustable centers play a key role when the number of hidden nodes is reduced to fewer than the number of training patterns since now, it is no longer possible to have a one-to-one correspondence between the training patterns and the centers of the RBF functions. Fig. 11 demonstrates the results of a RBF-SNLM mapping using fixed centers for the number of hidden nodes equal to one half the number of training patterns. Again, the red dots indicate the mapping of the points in the training set and the blue lines represent the connected interpolated points. Note that with fixed centers, there is some error with the mapping as expected. However, it is interesting to note that the interpolation of the points is relatively robust despite the relatively poor quality of the mapping for the training set. That is, they are still

mapped to the square curve. Fig. 12 illustrates the results with adjustable centers. Note that both isometric preservation and interpolation are greatly improved. We conclude, then, that incorporation of adjustable spread parameters and reducing the number of hidden nodes is constructive for interpolation. However, having fixed spreads with the same number of hidden nodes as training patterns results in the most accurate mapping of points in the original dataset. Fixed spreads, nonetheless, detract from mapping and interpolation when the number of hidden nodes is reduced.

4.2 *Congressional Voting*

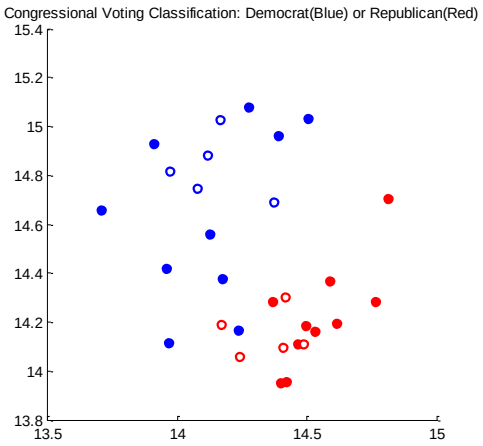

Figure 13. RBF-SNLM performed to differentiate Republicans from Democrats and interpolate unseen voting behaviors. Interpolated points are hollow. *Specifics*: 10 training patterns for each of Democrats and Republicans (20 total) were used. $H = N = 20$. 50 iterations were run via quasi-Newton Method. Adjustable weights and spreads were used. Dissimilarity between training patterns was calculated as a Tanimoto distance and normalized so that the maximum element was 1: only dissimilarities were used. Weights and spreads were randomly initialized in $(0, 1)$. An all-ones adjacency matrix was used. Centers were initialized via k-medoids. 5 additional patterns for each party (10 total) were interpolated.

Next, we apply RBF-SNLM in an attempt to visually discriminate between Democrats and Republicans based on their voting records. This particular dataset originates from the *1984 United States Congressional Voting Records Database* and represents the voting records of 435 House of Representative Members on 16 key issues as identified by a Certified Quality Auditor (CQA). The various issues range from immigration to education to handicapped infants. Votes were classified as one of three types, yes (y) (which include such key words as "voted for," "announced for" and "paired for"), no (n)(which include such key words as "voted against," "announced against" and "paired against"), and unknown $(?)$(which include "voted present," "voted present to avoid conflict of interest" and "did not vote or otherwise make a position known"). Note that the unknown classification does not necessarily mean missing data[24,25]. A sample training pattern is given as follows:

republican,n,y,n,y,y,y,n,n,n,y,?,y,y,y,n,y.

The dissimilarity between two training patterns is determined by the *Tanimoto distance*. We do not use the Euclidean distance here since it cannot be computed for the non-metric data. The Tanimoto metric is a simple and often successful concept used to calculate dissimilarites between two patterns that are judged to be either "same" or "different." It does not take into account graded similarities. Formally, $D_{Tanimoto}(P_1, P_2) = \frac{n_1+n_2-2n_{12}}{n_1+n_2-n_{12}}$, where n_i refers to the number of elements in set i and n_{ij} refers to the number of elements common to both sets i and j[26].

Fig. 13 depicts the results of the mapping of the Democrat and Republican voting patterns in blue and red respectively. There is a clear separation between the two parties and the interpolation of the data is overall accurate despite some error in isometric preservation. Note, however, that there is specifically one Democrat

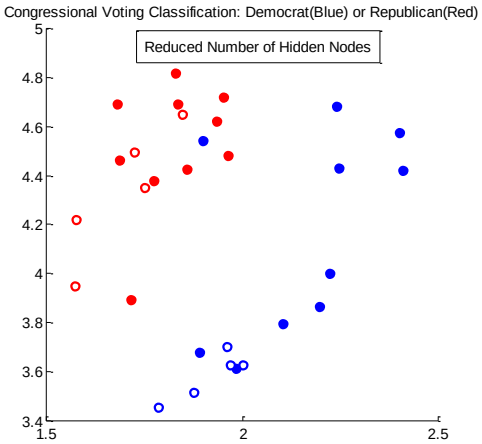

Figure 14. RBF-SNLM performed to differentiate Republicans from Democrats and interpolate unseen voting behaviors with a reduced number of hidden nodes. Interpolated points are hollow. *Specifics*: 10 training patterns for each of Democrats and Republicans (20 total) were used. $H = \frac{N}{2} = 10$. 50 iterations were run via quasi-Newton Method. Adjustable weights and spreads were used. Dissimilarity between training patterns was calculated as a Tanimoto distance and normalized so that the maximum element was 1: only dissimilarities were used. Weights and spreads were randomly initialized in $(0, 1)$. An all-ones adjacency matrix was used. Centers were initialized via k-medoids. 5 additional patterns for each party (10 total) were interpolated.

sample that is mapped close to the opposing party (Senator # 7). However, further examination of the dataset indicates that the two closest voting patterns to that particular senator in terms of Tanimoto distance are both that of Republicans'. Such instances reflect the real world phenomenon that voting behavior does not always conform with party affiliation.

Fig. 14 depicts the same results with a reduced number of hidden nodes. Again, note here that with a decreased number of hidden nodes, there is an even wider separation between the Democrats and the Republicans, thus resulting in an enhancement of visual discrimination between the two classes of senators. Note, however, that the outlying Democrat senator remains for the above reason.

4.3 *Federalist Papers*

The next dataset we examine consists of the series of *Federalist Papers*, which were written in 1787 and 1788 and published in various New York State newspapers with the aim of persuading the New York voters to ratify the United States Constitution. All essays were signed under the pen name *PUBLIUS*, but it is generally acknowledged that out of the collection of papers, Alexander Hamilton wrote 56, James Madison wrote at least 50, and John Jay wrote 5 papers[27]. There are 12 other disputed papers that are generally attributed to Madison[28,29]. The dataset provided contains 51 Hamilton paper patterns, 14 Madison paper patterns, and 12 disputed paper patterns. Each pattern contains the index of the specific Federalist Paper, the text file from which the data is extracted, and the frequencies (occurrences per 1000 words) of 18 key patterns of word phrases used in the particular paper[30].

We attempt to classify the papers in two ways. The first involves obtaining 14 patterns of each of the known Hamilton and Madison papers along with the 12 disputed papers. We then apply RBF-SNLM to the three groups of papers. We use an adjacency matrix, where preservation of all inter-cluster distances is considered important and preservation of all intra-cluster distances is ignored. Although there are more than 14 instances of Hamilton papers, we are limited by the number of Madison papers provided. Since the error function places more weight on a particular sample group when there are more patterns in it due to the non-zero entries in the adjacency matrix, we choose only the first 14 Hamilton patterns. We are, however, only limited to 12 disputed papers and select to use all of them. For the calculation of dissimilarities from the non-metric data, we deviate from the standard Euclidean distances to L_1 norms as they are slightly computationally simpler. Since the data

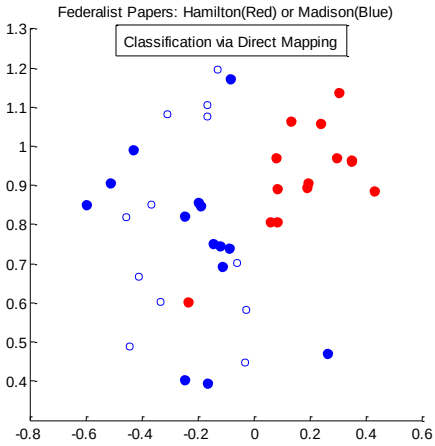

Figure 15. RBF-SNLM performed to identify the author of the disputed papers via direct mapping. Interpolated points are hollow blue. *Specifics*: 14 training patterns of each of known Hamilton and Madison papers and 12 training patterns of disputed papers (40 total) were used. $H = N = 40$. 150 iterations were run via quasi-Newton Method. Adjustable weights and spreads were used and initialized randomly in $(0, \delta_{max})$, for δ_{max} denoting the maximum dissimilarity between any two training patterns. Dissimilarity between two training patterns was calculated as L_1 distances (RBF-SNLM run on dissimilarities). An adjacency matrix that only preserves inter-cluster distances among all three categories was used. Centers were initialized via k-medoids.

provided reflects frequency counts of words, calculation of the L_2 norms does not give an explicit advantage over calculation of the L_1 norms. Since there is no need to perform additional interpolation on the dataset, the number of hidden nodes is kept to be equal to the number of training patterns for higher accuracy. The result can be seen in Fig. 15. Note that there is a relatively clear separation between the Hamilton and Madison patterns. Note also that in accord with previous theory, the disputed papers are identified to be visually closer to the Madison patterns.

4.3.1 Classification via Interpolation

As an alternate method, we also explore the interpolation of the disputed data patterns. First, via RBF-SNLM we use the "inter" adjacency matrix again with 14 of each of Hamilton and Madison training patterns (28 total) and L_1 norms for calculating dissimilarities to create a mapping in 2D. This adjacency matrix, as mentioned previously, considers preservation of all inter-cluster distances and ignores preservation of all intra-cluster distances. Since interpolation is involved, we choose the number of hidden nodes to be smaller than the total number of training patterns. After learning the mapping, we interpolate the disputed papers. Fig. 16 indicates that although the separation between the Hamilton and Madison patterns is rather small, the interpolated points still clearly fall on the Madison side. This mapping is consistent with previous results obtained via analysis performed using a support vector machine[29,31], where training was performed on 56 Hamilton papers and 50 Madison papers. Again, as shown in previous research, the separation between the two categories is not large[29].

4.4 *Swiss roll*

The *Swiss roll*[20] dataset, so called because of its shape, is a common test set used particularly for its distinct manifold and mapability. As its name suggests, it is a rolled up sheet in 3D. The aim here is to use RBF-SNLM to "unroll" the dataset so that it is mapped as a flat sheet in 2D with no overlapping points. Fig. 17 shows an image of the original dataset. Our particular roll consists of 5 equidistant layers with 30 points per layer for a total of 150 data points. In directly applying RBF-SNLM without interpolation of additional points, we work with the first three layers consisting of 90 data points unless otherwise stated.

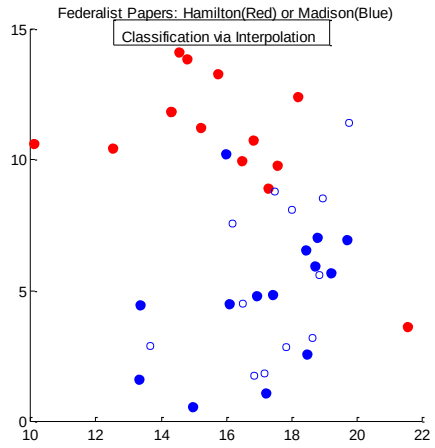

Figure 16. RBF-SNLM performed to identify the author of the disputed papers via direct mapping. Interpolated points are hollow blue. *Specifics*: 14 training patterns of each of known Hamilton and Madison papers (28 total) were used. $H = \frac{3N}{4} = (21)$: trade-off between accurate interpolation of previously unseen patterns and accurate mapping of the training set patterns. 200 iterations were run via quasi-Newton Method. Adjustable weights and spreads were used and initialized randomly in $(0, \delta_{max})$, for δ_{max} denoting the maximum dissimilarity between any two training patterns. Dissimilarity between two training patterns was calculated as L_1 distances (RBF-SNLM run on dissimilarities). An adjacency matrix that only preserves inter-cluster distances among all three categories was used. Centers were initialized via k-medoids.

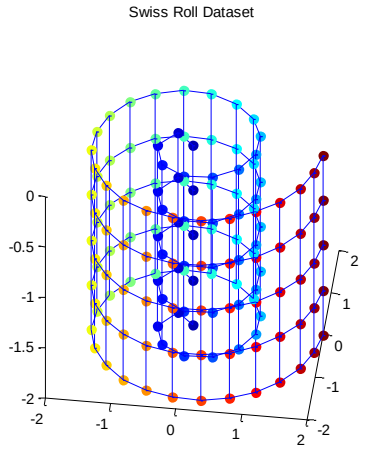

Figure 17. The *Swiss roll* dataset consisting of 5 equidistant layers and 30 points per layer.

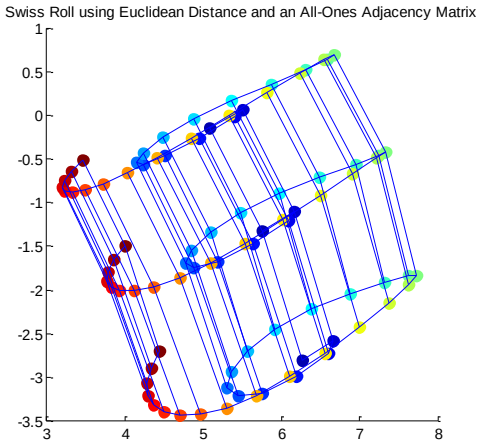

Figure 18. RBF-SNLM performed on the *Swiss roll* using Euclidean distance and an all-ones matrix. Blue lines indicate adjacencies between points in the original manifold. *Specifics*: 80 Iterations were run via quasi-Newton Method. Weights and spreads were initialized randomly. Centers were initialized as the training patterns. $H = N = 90$.

Fig. 18 depicts the results obtained from a generic mapping of the RBF-SNLM using Euclidean distances and an all-ones matrix. That is, there is no assumption of prior knowledge about the particular dataset. Note that here, the assumption of an all-ones matrix fails to unroll the *Swiss roll* and the points remain curled up as before.

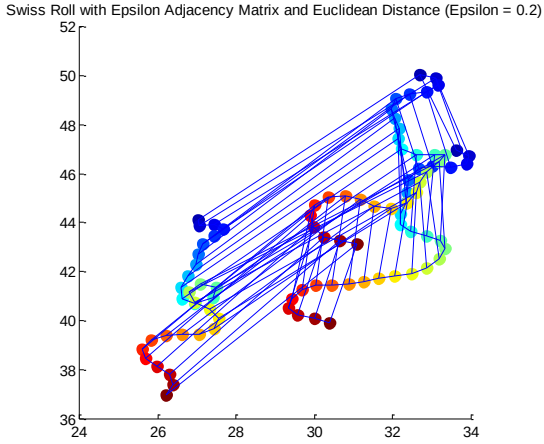

Figure 19. RBF-SNLM performed on the *Swiss roll* using Euclidean distance and an ϵ matrix with $\epsilon = 0.2$. Blue lines indicate adjacencies in original manifold. *Specifics*: 100 Iterations were run via quasi-Newton Method. Weights and spreads were randomly initialized. Centers were initialized as the training patterns. $H = N = 90$.

Previous research suggest a variation on the adjacency matrix for improved results[20]. Since we wish to unfold the dataset, we may place significantly less influence on preserving the distances between two points on a diameter of the trunk of the *Swiss roll*. We turn to the ϵ matrix as discussed in section 3.6 with $\epsilon = 0.2$ (20% of the maximum Euclidean distances between two patterns). Fig. 19 depicts the results. Note that in this particular case, while the *Swiss roll* is indeed unrolled, the resulting image does not accurately preserve the distances between the layers. One might think that increasing the ϵ value would yield a better mapping. Nonetheless, doing so only results in a failure to unroll the *Swiss roll*.

A third variation that we attempted aims to calculate the distances between two training patterns in the original feature space in terms of a geodesic distance, the shortest distance between two points on a manifold[20] as opposed to an Euclidean (L_2) norm. Note that in this case, the geodesic distance between any two training patterns will

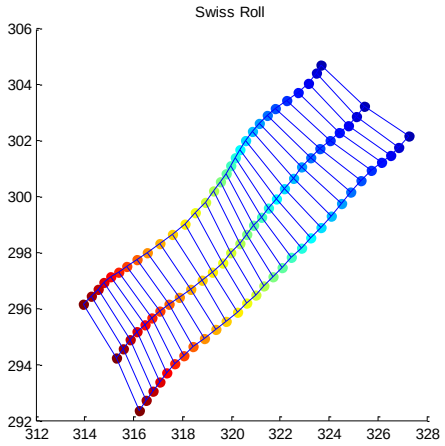

Figure 20. RBF-SNLM performed on the *Swiss roll* using geodesic distances and an all-ones matrix. Blue lines indicate adjacencies between two points in the original manifold. *Specifics*: 50 Iterations were run via quasi-Newton Method. Weights and spreads were randomly initialized. Centers were initialized as the training patterns. $H = N = 90$.

always be greater than or equal to the Euclidean distance between the same two patterns. Our manifold, is of course, the *Swiss roll* itself and graph distances are calculated via a shortest path algorithm such as Dijkstra's[32]. Note that the distances between training patterns in the target database are still calculated as Euclidean norms. Fig. 20 depicts the result using an adjacency matrix of all-ones. Note that in using this approach, we are able to unroll the dataset. Furthermore, the results can be achieved in a relatively short time (50 iterations) and the inter-layer distances are well mapped. Note, however, because of normalization of each particular dimension to be zero centered and have a standard deviation of one, the relative distances preserved between points are slightly distorted so that the resulting mapping is not a perfect square grid as it should be given equidistant adjacent points.

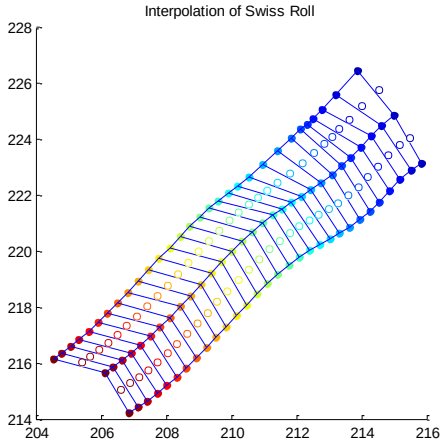

Figure 21. RBF-SNLM performed on the *Swiss roll* using geodesic distances and an all-ones matrix. Blue lines indicate graph connections between points. Hollow points indicate interpolated layers. *Specifics*: 100 Iterations were run via quasi-Newton Method. Weights and spreads were randomly initialized. Centers were initialized as the training patterns via k-Medoids. $H = \frac{N}{2} = 45$.

After successfully unrolling the *Swiss roll*, we aim also to interpolate intermediate layers. In training our patterns via direct RBF-SNLM mapping, we take layers 1, 3, and 5 of the *Swiss roll* and leave layers 2 and 4 for interpolation. Since we are interpolating, we reduce the number of hidden nodes to one half the number of

training patterns. Our results, shown in Fig. 21 indicate that interpolation is relatively accurate and the preservation of the inter-layer and intra-layer distances is faithful.

4.5 *Teapots*

Figure 22. Three frames of the *Teapots* dataset.

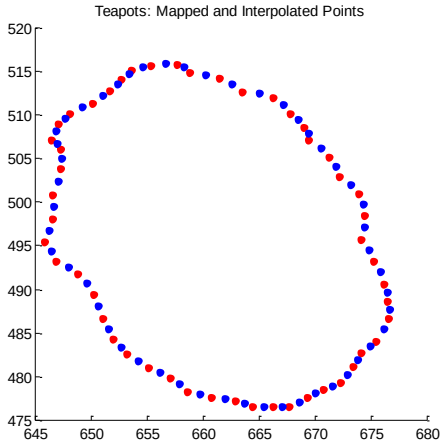

Figure 23. RBF-SNLM performed on the *Teapots* dataset using an arc length distance. Blue dots indicate mapped points. Red dots indicate interpolated points. *Specifics*: 100 iterations were run via quasi-Newton Method. Weights and spreads were randomly initialized. Centers were fixed to the training patterns. $H = N = 50$. An all-ones adjacency matrix was used.

The *Teapots* dataset[22,23] consists of 100 patterns each with 1938 features representing the grayscale values of predetermined pixels as the teapot undergoes a 360° rotation. Each image of the teapot is a 560 pixels by 420 pixels image sampled at every 3.6° angular rotation. Fig. 22 depicts three sample frames of the *Teapots* dataset.

For our particular experiment, we take every other pattern (grayscale frame) to form the training set. To find the dissimilarity between two points in the original database, we then calculate an arc length distance by applying the following rule. For each pattern in the training set, we find the two points (frames) that are closest in Euclidean distance and assign the dissimilarities between each of the two nearest neighbor patterns and that pattern an arc length distance of 2. We calculate the dissimilarities between any two points as a geodesic arc length distance. We then perform RBF-SNLM on this particular dataset. For accuracy of mapping and since we are interpolating on the manifold of the teapot curve, we do not reduce the number of hidden nodes.

For interpolation, we use the remaining patterns from our dataset. For each interpolated pattern, we also calculate its two nearest neighbors and assign that dissimilarity an arc length distance of 1. We then calculate the dissimilarities between a pattern and a center as a geodesic arc length distance as before.

The result we obtain from *Teapots*, as depicted in Fig. 23, indicates that this mapping is a rather smooth closed curve as expected, since the rotation of the teapot should place the resulting frames on a one-dimension, closed curve in 1938 dimensions. We also note that the interpolated points fit very well where they are supposed to.

5 CONCLUSIONS

Our work has introduced a variation of SNLM, a celebrated, metric Multi-Dimensional Scaling (MDS) technique to visualize data by attempting to preserve distances while non-linearly projecting the data to 2D or 3D. The new approach, referred to as RBF-SNLM, uses a Radial Basis Function (RBF) neural network to learn the Sammon map either by having full knowledge of the training patterns' location in the input feature space or by knowing only their pair-wise distances (or dissimilarities, in the general case). The key features of RBF-SNLM include an adjustable hidden layer complexity through the number of hidden units employed in the network hidden layer and adaptable spread and center parameters for each RBF. This flexibility allows for accurate mapping and interpolation of data from a higher dimensional space to a lower dimensional target space. In particular, we emerge with two versions of RBF-SNLM: the first of which performs the mapping knowing the exact locations of patterns in the input feature space, and the second of which performs the mapping knowing only the dissimilarity between each pair of training patterns. The second version has the advantage of visualizing non-metric data.

Moreover, we have made use of an adjacency matrix so that certain pairwise distances are given more emphasis to be preserved than others. With prior knowledge regarding the dataset, we are able to improve the visualization by appropriately structuring this adjacency matrix. Furthermore, zero entries in the adjacency matrix speed up calculation of the gradient and reduce computational time per iteration.

In addition, we have explored some variations in the choice of dissimilarities in the source database in order to suitably interpret the relationships between input patterns. Apart from the usual L_2 Euclidean Norm we have also explored the use of the L_1 Norm and the Tanimoto Distance for non-metric data. We concluded that using an appropriately defined dissimilarity may make interpretation of the available data more straightforward and meaningful.

Additionally, we have used RBF-SNLM for mapping interpolations, where the mapping is learned using an available training set and the obtained $\mathbf{W}, \mathbf{s}, \mathbf{C}$ parameters are used again to map previously unseen patterns into the target space. There is a trade-off between the accuracy of interpolation and the accuracy of mapping with respect to the training set. As seen from the results, a reduced number of hidden nodes will generally result in better interpolation. However, because of the reduced number of centers in the hidden layer, there is more inaccuracy in the exact preservation of dissimilarities calculated on the training data. A reduced number of hidden nodes also aids in providing a clearer separation between two categories of patterns, as demonstrated in the *Congressional Voting* dataset. Since we can achieve this separation via distance preservation, RBF-SNLM can be used as a visual classifier as it is done for the *Federalist Papers* dataset. The visual classification can be achieved in one of two ways: via directly applying RBF-SNLM and mapping the training data and also via interpolation of additional patterns once a mapping is learned. Finally, as demonstrated with the *Teapots* dataset, we expand our use of RBF-SNLM from metric to topological mappings, an aspect that we hope to further explore in the future.

Another path to explore is mapping to the hidden layer via a different kernel function. Our current mapped SNLM approach uses a Gaussian RBF since it is relatively common and relatively easy to specialize to Classical-SNLM by adjusting the spread and center parameters. In specific, we could investigate the effect of using different RBFs to the visualization results obtained via RBF-SNLM and assess the interpretability of the projected data.

Finally, for the immediate future, we would like to explore techniques of adjusting the centers when we are only cognizant of pair-wise dissimilarities between the available data. While one route suggests combinatorial optimization techniques to identify training patterns as RBF centers, another line of research could be developed by investigating inner product based techniques to express RBF centers as convex combinations of training patterns.

A DESIGN OF IMPLEMENTATION

One of the challenging aspects of the project was creating a flexible, extensible, object-oriented design of the SNLM implementation. The design had to be generic in order to allow for easy changes to key components, such as the line search or the optimization method. Furthermore, it needed to be able to perform Classical-SNLM and RBF-SNLM, with either the training patterns or without exact knowledge of the training pattern locations and knowledge only of pair-wise dissimilarities between patterns.

In the design of SNLM, we have three main classes, the *Knowledge Representation*, the *Training Algorithm*, and *Test*. These are shown in Fig. 24. The *Knowledge Representation* contains all of the parameters of the model necessary to allow another implementation to reconstruct the target and perform interpolation/extrapolation. The *Knowledge Representation* contains the weights, the spreads, and the centers parameters. It also contains a reference to the type of *Radial Basis Function* the model uses.

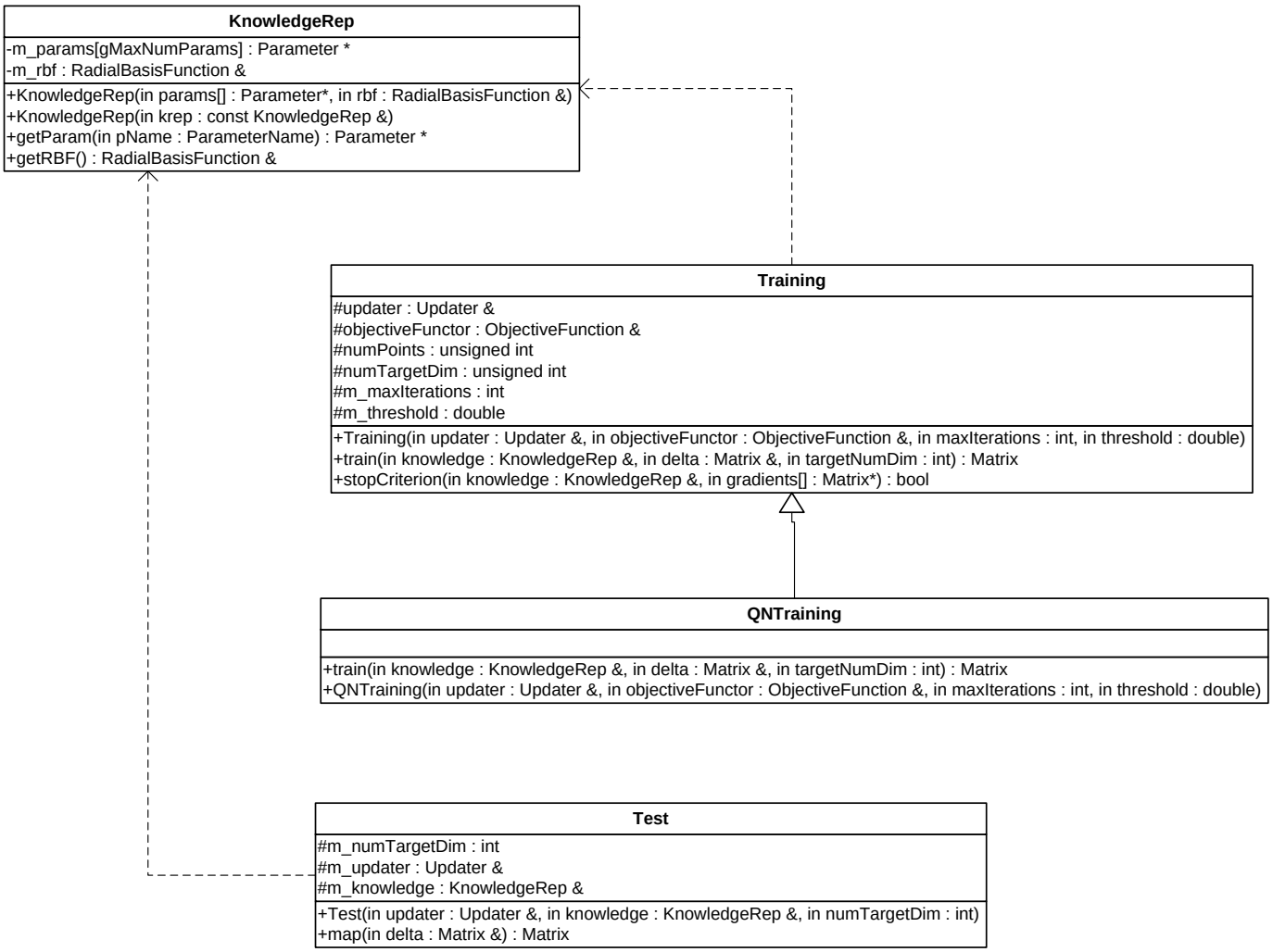

Figure 24. Class diagram illustrating the relationships of the main components. The *Training Algorithms* are highly dependent on the *Knowledge Representation* for what behavior they produce. There is an inheritance relationship here that produces *QNTraining*. The *Test* algorithm's output is dependent on a previously trained model's *Knowledge Representation*.

A.1 *Training Algorithm*

The *Training Algorithm* takes a *Knowledge Representation* containing all of the initialized parameters and trains this model so that the stress function in Eq. 3.4 is minimized.

The *Train* method implements the psuedocode of Alg. 1. There is a key difference though: the *Train* method

updates $\mathbf{\Phi}$, $\mathbf{Y}$, and $\mathbf{D}$ with an Updater function object, discussed in section A.9. This, in addition to the non-specific parameter optimization loop, allows the *Training Algorithm* to be generic enough to be used in multiple situations where the parameters, RBF, or stress function may all be different from what they are in RBF-SNLM. For instance, we can produce Classical-SNLM when the *Knowledge Representation* contains only the weights and the RBF is the one in Eq. 3.2. The algorithm uses a stopping criterion method that checks if the stopping criterion is met.

Although the base class *Training* is generic enough to satisfy many different configurations of parameters, line searches, and optimization algorithms, specialized handling code is sometimes necessary in the training. If this is necessary, then the *Train* method is overridden in a derived class.

The *Quasi-Newton Training (QNTraining)* class is derived from the base class Training. It contains specialized code for handling failures of the line searches when using *BFGS Quasi-Newton* as the optimization algorithm. If a line search fails to find a step length satisfying the Wolfe conditions, then the parameter currently undergoing adaptation is not adapted and the *BFGS Quasi-Newton* optimization algorithm is reset to use the identity matrix as the previous Hessian matrix, effectively becoming gradient descent.

A.2 *Test*

The *Test* class performs interpolation and extrapolation. It takes a set of test patterns and a *Knowledge Representation* object from a previously trained model and maps these test patterns to the target space. To do this, it constructs an interpolation matrix $\mathbf{\Phi}$ from the test patterns and *Knowledge Representation* object and uses Eq. 2.6 to find the mapped outputs. The *Test* class uses an *Updater* object to construct the interpolation matrix so that the class is generic to any *Parameters*.

A.3 *Knowledge Representation*

The *Knowledge Representation* is a container class in that it stores all *Parameters* in a collection and exports the getParam method as an accessor and mutator for each *Parameter* object. A *Knowledge Representation* object need not know about what parameters it stores and thus can be generic to all parameter sets. The *Knowledge Representation* also stores a reference to the RBF being used as this is crucial to the successful operation of the *Test* class.

A.4 Parameters

All values for a parameter in an SNLM model are stored in a *Parameter* container object, shown in Fig. 25. In addition, the *Parameter* class contains a reference to the stress function's gradient with respect to the parameter type. *Parameter* also store references to the *Line Search* and *Optimization Algorithm* to use with them. A *Parameter* can be adaptable or non-adaptable. There should be one instance for each parameter type: the weights, spreads, centers, and so forth.

Semantically, it may make more sense to have the *Gradients* referenced in the *Objective Function*. However, one of the goals was to reduce the amount of code dependency, so as to allow for flexibility with which components to use. The semantically correct way forces specialized code to be written in the *Line Searches* which handle which parameter they are passing to the gradient in the *Objective Function*. Thus, we assign the *Gradient* reference to its respective parameter.

Another design choice was the inclusion of the *Line Search* and *Optimization Method* in the *Parameter* container. This is because the *Line Searches* and *Optimization Methods* cache data between private method calls, such as the directional derivative at some step length α. These make the classes very parameter-specific. It was natural to group them with the *Parameter* class. A side benefit to grouping these classes this way is that each *Parameter* can have a different *Line Search* and *Optimization Method*.

A.5 *Radial Basis Function*

The *Radial Basis Function* is a functor class that evaluates an RBF given a distance and constant. All *Radial Basis Functions* are derived from the same interface which takes the distance and an extra constant whose meaning depends on the RBF type.

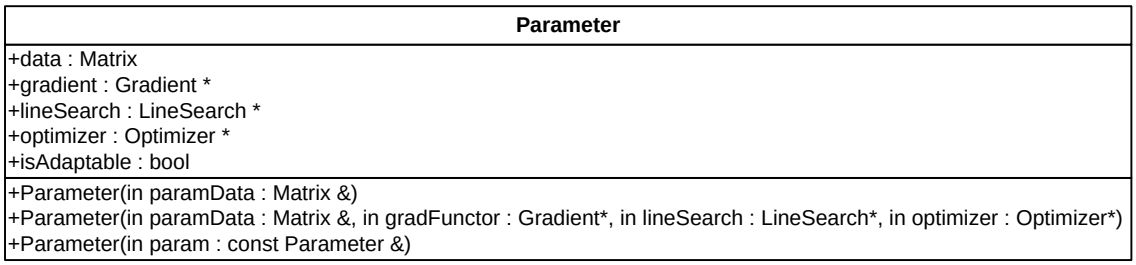

Figure 25. Class diagram showing that the parameter class is a container class for the parameter data and parameter specific functors.

A.6 *Objective Function*

Another class is the *Objective Function*, used by the *Line Searches* to test trial step lengths' costs. The *Objective Function* can be called as a function to evaluate a current configuration of the *Knowledge Representation* or a hypothetical one if a step length α is used to adapt a particular parameter. It can also find the directional derivative of a hypothetical *Knowledge Representation* configuration. The *Objective Function* uses an abstract base class which must be inherited to have a standard interface. The *Objective Function* used in RBF-SNLM is the *Sum Of Squares Error* function.

A.7 *Gradients*

The *Gradient* classes, shown in Fig. 26, are functors referenced by the *Parameter* class which calculate the gradient of an *Objective Function* with respect to that parameter. These all derive from the same interface.

There are two types of *Gradient* classes: *Gradients* with the original training patterns and *Gradients* using only pair-wise dissimilarities. Both derive from an abstract base class. Each *Gradient* calculates intermediate terms differently depending on its base type. If the *Gradient* must have access to the original patterns, such as the gradient of the stress function with respect to the centers, then that *Gradient* class must be derived from the appropriate abstract base class so as to standardize the constructors. The *Gradient* methods are the most computationally intensive in the implementation. All efforts to optimize them should be taken.

A.8 *Line Search* and *Optimization Method*

The *Line Search* class determines the adaptation step length. From its interface are derived *Constant Step Length*, *Backtracking Line Search*, and *Wolfe (conditions based) Line Search*. The *Optimization Method* class determines the search direction. From its interface are derived *Gradient Descent*, *Conjugate Gradient Descent*, and *BFGS Quasi-Newton* method.

A.9 *Updater*

The *Updater* class returns the new auxiliary matrices $\mathbf{\Phi}$, $\mathbf{Y}$, and $\mathbf{D}$ based on the configuration of a *Knowledge Representation* object passed to it. The *Updater* is specific to the type of SNLM being performed. For instance, there is a different one for Classical-SNLM than for RBF-SNLM because the parameters used are different.

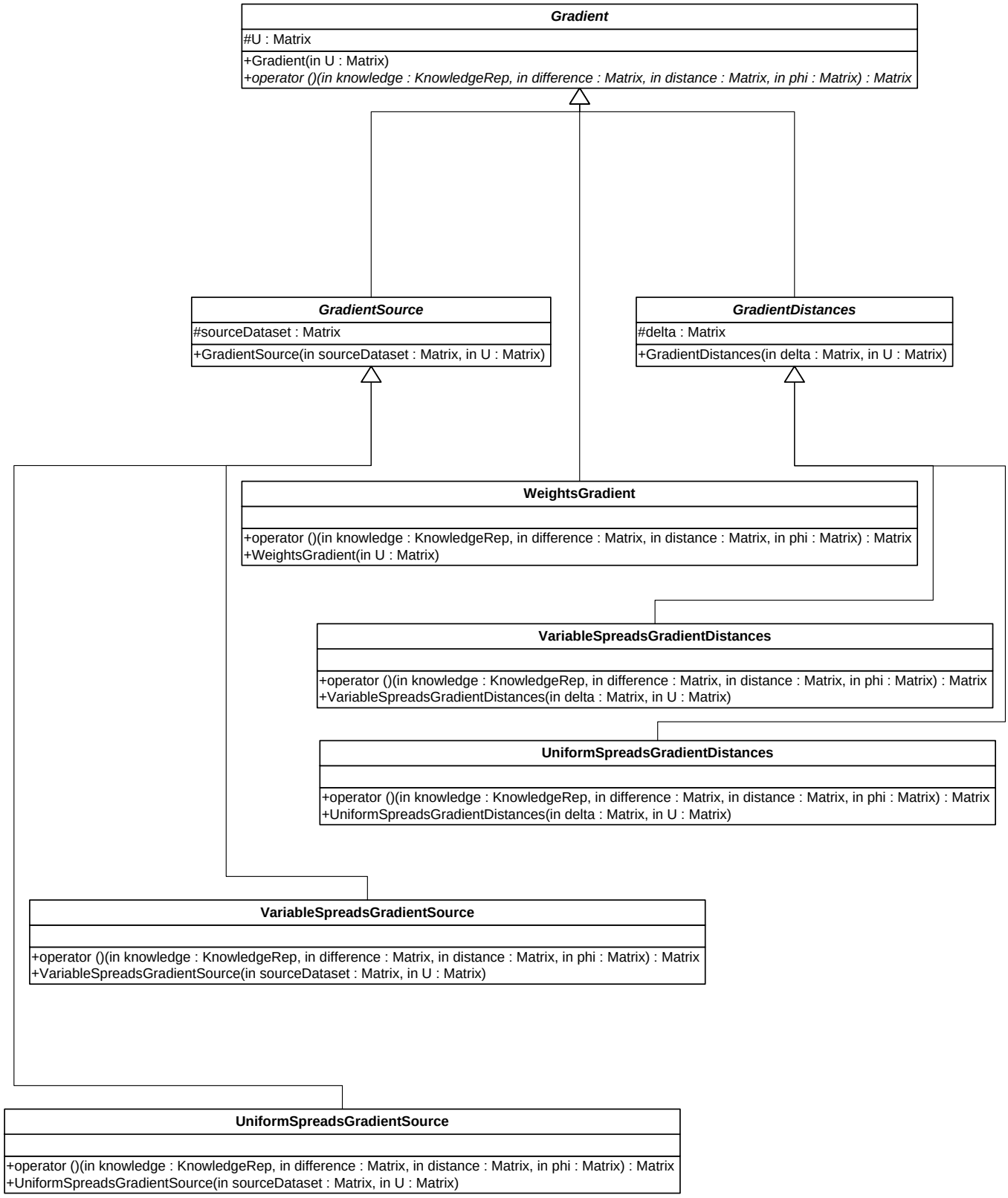

Figure 26. Class diagram showing that all gradients derive from the abstract base class gradient so that they can be used polymorphically.

ACKNOWLEDGMENTS

This material is based upon work/research supported in part by the National Science Foundation under Grant No. 0647120 and Grant No. 0647018. Mingbo Ma and Georgios C. Anagnostopoulos also acknowledge partial support from National Science Foundation Grant No. 0717674. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Sammon, J. W., "A nonlinear mapping for data structure analysis," *IEEE Trans. Comput* **C**(18) (1969).

[2] Young, G. and Householder, A. S., "Discussion of a set of points in terms of their mutual distances," *Psychometrika* **3**, 19–22 (1938).

[3] Torgerson, W. S., "Multidimensional scaling, i: theory and method," *Psychometrika* **17**(4), 401–419 (1952).

[4] Haykin, S., [*Neural Networks: A Comprehensive Foundation*], Prentice Hall, Upper Saddle River, New Jersey (1999).

[5] Powell, M. J. D., "Radial basis functions for multivariable interpolation," in [*IMA Conference on Algorithms for the Approximation of Functions and Data*], 143–167 (1985).

[6] Light, W., "Ridge functions, sigmoidal functions and neural networks," in [*In Approximation Theory VII*], 163–206, Academic Press (1993).

[7] Park, J. and Sandberg, I., "Universal approximation using radial-basis-function networks," *Neural Computation* **3**(2), 246–257 (1991).

[8] Cox, T. F. and Cox, M. A., [*Multidimensional Scaling*], Chapman and Hall/CRC, Boca Raton (2001).

[9] Haykin, S., [*Neural Networks: A Comprehensive Foundation*], Prentice Hall, Upper Saddle River, New Jersey (1999).

[10] Mao, J. and Jain, A. K., "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Transactions on Neural Networks* **6**(2), 296317 (1995).

[11] de Ridder, D. and Duin, R. P. W., "Sammon's mapping using neural networks: A comparison," *Pattern Recognition Letters* **18**(1113), 13071316 (1997).

[12] Hornik, K., Stinchcombe, M., and White, H., "Multilayer feedforward networks are universal approximators," *Neural Networks* **2**, 359–366 (1989).

[13] Trenn, S., "Multilayer perceptrons: Approximation order and necessary number of hidden units," *IEEE Transactions on Neural Networks* **19**, 836–844 (May 2008).

[14] Micchelli, C. A., "Interpolation of scattered data: Distance matrices and conditionally positive definite functions," *Constructive Approximations 2* **2**(1) (1986).

[15] Nocedal, J. and Wright, S. J., [*Numerical Optimization*], Springer-Verlag New York, Inc., New York, NY (1999).

[16] Broyden, C. G., "The convergence of a class of double-rank minimization algorithms," *IMA Journal of the Institute of Mathematics and Its Applications* **6**, 76–90 (1970).

[17] Fletcher, R., "A new approach to variable metric algorithms," *Computer Journal* **13**, 317–322 (1970).

[18] Goldfarb, D., "A family of variable metric updates derived by variational means," *Mathematics of Computation* **24**, 23–26 (1970).

[19] Shanno, D. F., "Conditioning of quasi-newton methods for function minimization," *Mathematics of Computation* **24**, 647–656 (1970).

[20] Lee, J. A. and Verleysen, M., [*Nonlinear Dimensionality Reduction*], Springer, New York, NY (2007).

[21] Kaufman, L. and Rousseeuw, P. J., [*Finding Groups in Data: An Introduction to Cluster Analysis*], Wiley-Interscience, 9 ed. (March 1990).

[22] J.B. Tenenbaum, V. d. S. and Langford, J., "A global geometric framework for nonlinear dimensionality reduction," *Science* **290**, 2319–2323 (December 2000).

[23] Tenenbaum, J., "Mapping a manifold of perceptual observations," *Advances in Neural Information Processing Systems (NIPS 1997)* **10**, 682–688 (1998).

[24] Schlimmer, J. C., *Concept acquisition through representational adjustment*, PhD thesis, University of California, Irvine (April 1987).

[25] "Congressional quarterly almanac, 98th congress, 2nd session 1984," (1985).

[26] R. O. Duda, P. E. Hart, D. G. S., [*Pattern Classification*], John Wiley and Sons, INC., New York, NY (2001).

[27] Whitten, C., "The federalist papers." Founding Fathers. Series: Founding Fathers Info. http://www.foundingfathers.info/federalistpapers/ (2007).

[28] "The federalist papers." Founding Fathers. http://www.foundingfathers.info/federalistpapers (2007).

[29] Fung, G., "The disputed federalist papers: Svm feature selection via concave minimization," *Journal of the ACM* **V**(N), 1–9 (20YY).

[30] Asuncion, A. and Newman, D., "UCI machine learning repository." http://www.ics.uci.edu/~mlearn/MLRepository.html (2007).

[31] Bosch, R. and Smith, J. A., "Separating hyperplanes and the authorship of the disputed federalist papers," . *American Mathematical Monthly* **105, 7**, 601–608 (1998).

[32] Djikstra, E. W., "A note on two problems in connexion with graphs," *Numerische Mathematik* **1**, 269–271 (1959).