DISSERTATION

LOCAL NONLINEAR MODELING VIA NEURAL CHARTS

Submitted by

Douglas Robert Hundley

Department of Mathematics

In partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 1998

COLORADO STATE UNIVERSITY

June 12, 1998

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY DOUGLAS ROBERT HUNDLEY ENTITLED "LOCAL NONLINEAR MODELING VIA NEURAL CHARTS" BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Department Head

ABSTRACT OF DISSERTATION

LOCAL NONLINEAR MODELING VIA NEURAL CHARTS

We present an approach for the representation of dynamical models via the construction of *neural charts*, which provide an optimal local coordinate system for both data reduction and semi-analytic remodeling problems. The chart functions are local linear mappings while their inverses are nonlinear and are approximated by neural networks. This technique therefore represents an improvement over local linear methods due to the nonlinear reconstruction; as a result, the density of the clusters is significantly lower and the overall representation more accurate. The compressed dynamics are integrated locally (traditionally and via neural models) in spaces of reduced dimension and the trajectories accurately reconstructed. We demonstrate the method using the Lorenz attractor, a torus, and several examples from the Kuramoto-Sivashinsky equation.

Douglas Robert Hundley
Department of Mathematics
Colorado State University
Fort Collins, Colorado 80523
Summer 1998

ACKNOWLEDGEMENTS

First of all, I owe a deep dept of gratitude to my advisor, Michael Kirby, without whose generosity and unending stream of ideas this dissertation would not have been possible.

Secondly, this work would have been impossible without the support of my family.

I would also like to thank Mark Muldoon, Dave Broomhead and Greg King for their many useful discussions that were critical to the development of the work presented in Chapter 3.

Finally, thanks to Glaxo Virtual Anatomy Project for supplying the lung data used in Chapter 4.

This research was supported in part by the NSF under grants ECS-9312092, INT-9513880 and DMS-9505863, and by a Ph.D. Dissertation Fellowship awarded by Colorado State University (1997-98).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

There are a number of important scientific and technological problems which involve the simulation of high-dimensional dynamical systems with global attractors. It is now well established that in many cases the high-dimensionality of a dynamical model is not a true reflection of the intrinsic dimensionality of the globally attracting set on which the long term dynamics reside. Computational evidence of low-dimensional dynamics has been provided by global reduction techniques including the nonlinear Galerkin [44, 55, 39, 65, 71, 15], the linear Karhunen-Loève (KL) [12, 66, 68] and the constrained nonlinear Karhunen-Loève (CNLKL) procedure [48].

This dissertation is concerned with similar models: simulations of dynamical flows (in high dimensional space) for which there exists an attractor that can be approximated by a manifold, or more generally, embedded in a manifold. We seek to establish local semi-analytical representations for the dynamics which reflect the number of asymptotic degrees of freedom of the solution. In particular, we propose a method for approximating an invariant manifold which in turn is an approximation of the global attractor. This distinction (between the attracting set itself and a manifold which approximates it) is important in view of the universal approximation capability of smooth functions by artificial neural networks [35, 13], an integral component of our method.

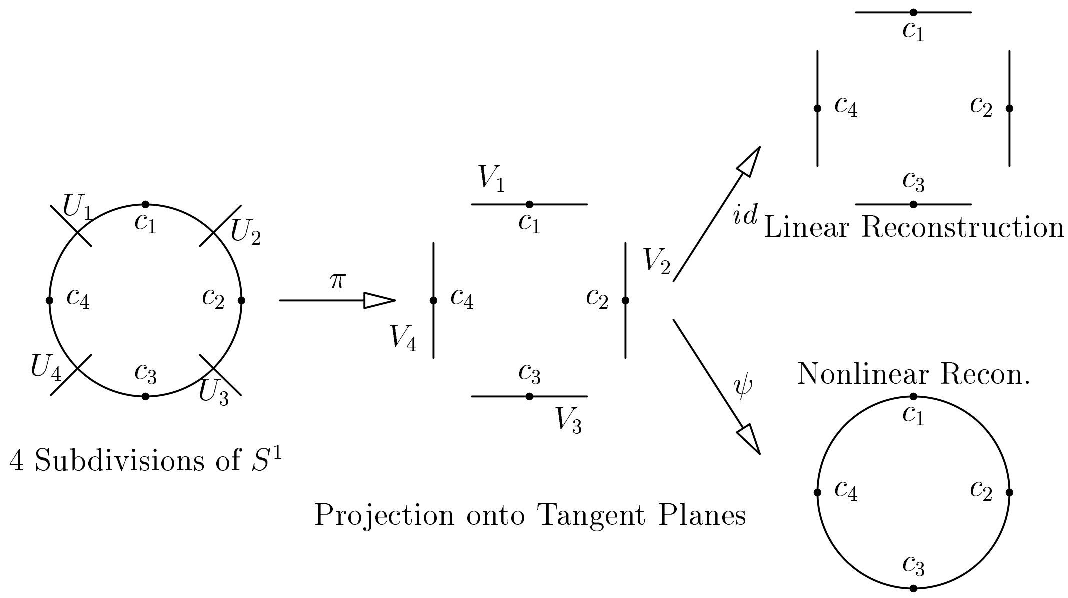

Figure 1.1: Local Reconstruction of the Circle: Nonlinear vs. Linear.

It is well known that while a manifold may be locally very low dimensional, it is simple to embed it in such a way as to be globally very high dimensional (i.e., imagine a knot embedded in 200-dimensional space). In general, global linear transformations will fail to reduce such curves optimally. However, it is possible to construct local linear mappings that are capable of reducing the dimension to reflect the intrinsic dimensionality simply by projecting the local region onto its tangent space. In order to properly define the tangent space, we assume that we can construct a local coordinate system so that the local manifold can be expressed in the form $(\boldsymbol{y}, f(\boldsymbol{y}))$, and the tangent space is then the span of the Jacobian of f at the cluster center. The associated linear reconstruction of the curve in the original space is then piecewise linear. As a result, a very large number of local regions may be required to approximate the original data to high accuracy. In general a nonlinear inverse function is essential to maintaining the accuracy of the model which is being reduced and will substantially reduce the number of regions one needs for a local approximation.

Consider the following simple example that illustrates this point. Suppose a given data set consists of points on a circle, i.e., a one dimensional manifold which is globally 2-dimensional (see Figure 1). Now choose several local one dimensional regions appropriate for local approximation (so that the projections are $1-1$). For the purposes of this illustration, we select four regions that divide the circle into equal subsets. In Figure 1, these subsets are U_1, U_2, U_3, U_4 with corresponding cluster centers c_1, c_2, c_3, c_4. Each region U_i may be projected onto its associated local tangent space V_i based at its cluster center c_i. The union of the four local projections is denoted collectively by π. The circle is now decomposed into four one dimensional sets, a representation with the same intrinsic dimensionality as S^1. As stated previously, however, the "inverses" of the projection maps construct only the local tangent spaces. In the picture, the one-dimensional subspaces are already shown as tangent spaces, so this inverse is pictured as the identity mapping, *id*. Also depicted in the figure is the mapping that would be obtained if the four true nonlinear inverses (collectively denoted ψ) were constructed. Clearly, if dynamics along the circle are simulated using the linear reconstruction, it would introduce large discontinuities (corresponding to jumps between the tangent planes) to the reconstructed flow in two dimensions (as will also be seen in the examples in the last part of this proposal). On the other hand, the nonlinear reconstruction of the flow should have almost no error and is clearly the desireable approach.

This is the conceptual outline of the new local, nonlinear, dimensionality reducing algorithm called *neural charts*. This approach divides the phase space into a collection of local regions, and then builds an array of dimensionality reducing (through the local linear projections) and reconstruction (semi-analytic, nonlinear) mappings, one for each region. The key part of the algorithm is to use neural networks to approximate the local nonlinear inverses to the local projections. Once these have been constructed, the algorithm will have produced a full local description of the manifold, and this description can be used as the basis for reconstructing the original dynamical system. This algorithm was first introduced in [37].

The results of the neural charts will be used to first compress, then remodel the original dynamical system. The following is a summary of this process.

Algorithm 1.1 *The Neural Charts Algorithm*

1. Cluster the data set X, obtaining cluster centers $C = \{\mathbf{c}_i\}_{i=1}^{p}$ using a clustering algorithm from those discussed in Chapter 2.
2. Partition X into sets U_i using the decision function constructed in Step 1.
3. Construct an approximation to the tangent space of U_i, and compute the linear projection maps π_i.
4. Form the neural reconstructions ψ_i.

Below we illustrate the neural charts algorithm and the dynamical reconstruction. Techniques for dynamical remodeling are discussed at length in the last chapter; here we summarize how the neural charts perform the remodeling. In general, the goal is to remodel the system of ordinary differential equations

$$\dot{\mathbf{x}}(t) = F(\mathbf{x}) \tag{1.1}$$

the trajectories of which are assumed to asymptotically lie on an (approximate) manifold, $\mathbf{x}(t) \in X \subset \boldsymbol{R}^N$. We have that locally,

$$\mathbf{v}(\mathbf{t}) = \pi_i(\mathbf{x}(t)$$

so that, by differentiating and substituting for $\dot{\mathbf{x}}$ the expression in Equation (1.1), we obtain the following local, low dimensional description for the dynamics:

$$\dot{\mathbf{v}}(t) = D\pi_i(\mathbf{x}(t)) \cdot F(\mathbf{x}(t))|_{\mathbf{x}(t)=\psi_i(\mathbf{v}(t))} \tag{1.2}$$

The approximate inverse ψ_i plays a crucial role in Equation (1.2) given that the vector field $\mathbf{x}(t)$ is actually computed from the nonlinear reconstruction. The Jacobian matrix, $D\pi_i$ is constant for the region V_i, and so if we let A_i be the M_i x N matrix formed by the basis vectors for U_i, equation (1.2) becomes

$$\dot{\mathbf{v}}(t) = A_i \cdot F(\psi_i(\mathbf{v}(t))) \tag{1.3}$$

The following algorithm was used to implement the local integration.

Algorithm 1.2 *The Local Modeling Algorithm*

1. Choose an initial $\mathbf{x}$.
2. Find the local chart for $\mathbf{x}$.
3. Project $\mathbf{x} \in U_i$ to $\mathbf{v} \in V_i$.
4. Perform one integration step on $\mathbf{v}$ (using equation (1.3) to find dv values) to obtain the new local value $\mathbf{v}$.
5. Perform neural reconstruction to $\mathbf{v}$, obtaining a new $\mathbf{x}$.
6. Repeat from step (2).

The purpose of this dissertation is to provide a detailed implementation of the neural charts algorithm, provide the background and relavent research, provide examples of how it can be used, extend the algorithm to more complex examples, and perform a comparison to other techniques.

In the process of determining an algorithmic method by which we can decompose the manifold, questions concerning pattern analysis and functional analysis have been researched.

Many of these questions are inherently ill-posed. For example, if we have noisy data, at what point does the noise become inextricably linked to the signal? And,

	Global Methods	Local Methods
Linear	Galerkin	Local Adaptive Galerkin VQPCA Wavelets
Nonlinear	Nonlinear Galerkin Inertial Manifolds Bottleneck NN	Neural Charts

Table 1.1: Conceptual Classification of Modeling Techniques. These are the methods used for the dimensionality reduction of dynamical systems.

while many of the objects we work with (i.e., the tangent surface to a manifold) are formally well defined, the definitions are difficult to translate to a numerical algorithm. Similarly, while we may understand heuristically what is meant by terms such as "dynamically homogenous regimes" and "coherent structures", how are these ideas translated down to a numerical level?

We present here the basic tools of pattern analysis and artificial intelligence that have been fundamental in the attempt to answer the questions posed above: data clustering, tangent space approximation, dimensionality reduction, and neural networks. Individually, these tools represent the cutting edge in pattern analysis, and each gives important information about certain aspects of the data. Together, they represent the basis of the neural charts algorithm.

To help put the neural charts algorithm into perspective, consider the general classifications given in Table (1.1). These methods are defined and described in depth in Chapter 7. Here we note the importance of the neural charts algorithm. It is uniquely positioned to take advantage of both local linear and nonlinear features of the manifold on which the dynamics under study reside, while other methods do not possess this flexibility.

In developing the neural charts algorithm, significant advances have been made in the three components to the algorithm. We have developed a powerful space-time clustering paradigm that can be incorporated into existing clustering algorithms that will be discussed in Chapter 2. We have developed a strategy by which the local dimension of the tangent plane can be determined, combining the ad-hoc criteria of Fukunaga with the scaling arguments of Broomhead and colleagues. This will be discussed at length in Chapter 3. Finally, we have applied the powerful function interpolation properties of neural networks to the problem of manifold reconstruction, which has not been previously attempted. This last step in the algorithm is key in the preservation of the true dynamics. The neural networks are discussed at length in Chapter 4, and Chapter 5 goes into detail about the implementation of the neural charts code. In Chapter 6 we put the neural charts algorithm into practice with some examples. Here we will see the failings of the manifold approximation with strictly linear maps. Furthermore, we have used neural networks in a data triangulation task, and show our results in Chapter 4.

Finally, in Chapter 7 we discuss the state of the art in dynamical reduction models, and compare the neural charts algorithm to these techniques.

Chapter 2

DATA CLUSTERING

In this chapter, we review the clustering algorithms currently in use. We will see that they share many common characteristics, including the assumption that the data set under study can be accurately clustered using spatial information alone. This leads us to consider a clustering algorithm designed specifically for attracting sets of dynamical systems, and culminates in the development of a space-time clustering algorithm.

2.1 Introduction

The process of clustering is to obtain a small set of points that best describes a given data set. Let this set of points be denoted by $\{\mathbf{c}_i\}_{i=1}^p$, and we will refer to these points as cluster *centers*. Denote the set of all cluster centers by C. The process of clustering can be thought of as finding a mapping which takes the data set X to a set of integers:

$$m : X \to \{1, 2, \ldots, p\}$$

so that, if $m(\boldsymbol{x}) = i$, then $\boldsymbol{x}$ is most closely represented by center $\boldsymbol{c}_i$. Thus, m is a decision function, and the clustering algorithm will be constructing m.

Throughout our discussion, we will be comparing a data point, $\boldsymbol{x}^{(\mu)}$ with its associated center, c_i, where $i = m(\boldsymbol{x}^{(\mu)})$. For shorthand, we will write

$$c_{m(\mu)} \equiv c_i \text{ where } i = m(\boldsymbol{x}^{(\mu)})$$

The purpose of the clustering algorithm is to represent the data with as few cluster centers as possible— in essence, reducing the data to a *locally* one dimensional data set. A global one dimensional reduction of the data would mean representing the data by its mean, and on the other extreme we could use as many centers as data points. Although the latter case is also a local one dimensional reduction where we have retained all the information contained in the data, there is no global compression. In the former case, we have excellent compression, but have retained almost no information about the original data. Rather, we seek a trade off between these extremes, where the set of centers represent (globally) a compression of data, and locally we have lost little information contained in the original data set.

Therefore, the choice of clustering routines depends on what features of the data are meant to be preserved, and this is expressed in terms of a measure of error which will be minimized by a certain placement of the centers. For example, certain data sets possess a natural classification based solely on their position in space, such as the data set in Figure 2.1.

We will see in this chapter that the algorithms currently in use (Kohonen's SOM, Neural Gas, VQPCA and LBG) all make the implicit assumption that there is a natural partitioning of the space extractable from only the data itself, and this assumption is expressed in the decision function, m. For the SOM, Neural Gas and LBG algorithms, the decision function simply takes the form:

$$m(\boldsymbol{x}) = i \Leftrightarrow \|\boldsymbol{x} - \boldsymbol{c}_i\|_2 \leq \|\boldsymbol{x} - \boldsymbol{c}_j\|_2, \; \forall i \neq j$$

This assumption is incorrect in many clustering tasks. Often, we have extra information that should be taken into account when forming the partition boundaries. In some settings, this information takes on the form of "class labels", where some of the data is labelled as being from class 1, class 2, etc. Many researchers

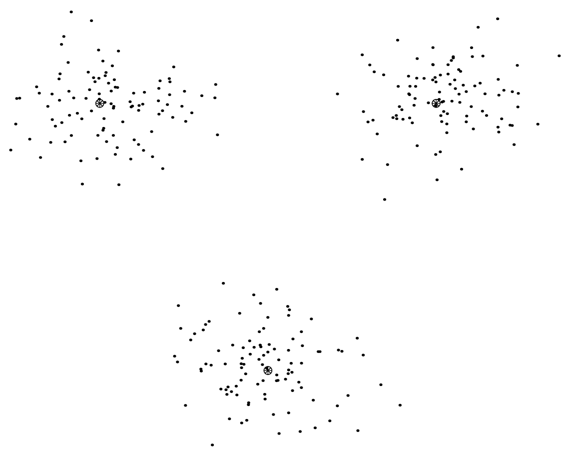

Figure 2.1: Clustering of three spatially distinct regions. Cluster centers are depicted as circles, and are the local means.

take this case separately, and form algorithms based on "supervised learning" (versus what we have described previously, which would be called "unsupervised learning"). This is not necessary, however. We will show, in constructing the Space Time LBG algorithm, that additional information can be included directly in the clustering algorithm by increasing the dimensionality of the data, incorporating a suitable metric on the new space, and modifying the objective function that is being minimized.

For example, the data we work with in the neural charts task is data derived from a dynamical system, and the derivative information is not only available, but as we will see later, it is often necessary to take this information into account when partitioning the data sets.

We will now present two "template" algorithms which help to distinguish between clustering methods currently in use. The clustering algorithms have two distinctly different flavors. The first template is called the *neural*, or soft, classifier. The main feature of this algorithm is it uses a type of annealing function to place the cluster centers. The second type of algorithm is based on the fact that, given a ball of data, its "best" representation is its mean. First, we discuss the soft clustering algorithm.

The idea behind this technique consists of setting up a competitive forcing between data points. That is, every data point will be attempting to attract all cluster centers to itself. For example, if the data consists of the unit circle, and we have one cluster center at the origin, then the cluster center would not move. The competing forces have set up an equilibrium at the circle's center. On the other extreme, if there is one data point and several clusters, all clusters will move to that data point.

In the algorithm that follows, each data point will have an attractive force that is radiating from itself in a linear fashion. Therefore, for a given center, $\boldsymbol{c}_i$, the attractive force is along the vector $\boldsymbol{x} - \boldsymbol{c}_i$

Algorithm 2.1 *Soft Clustering via a Vector Field: Type I Algorithm.*

We assume our data is given by X in $\mathbb{R}^N$.

1. Initialize the algorithm by randomly placing the cluster centers in $\mathbb{R}^N$.

2. Main Loop:

   (a) Choose data point $\boldsymbol{x} \in X$ at random.[1]

   (b) Update every cluster center's position:

$$\boldsymbol{c}_i(t+1) = \boldsymbol{c}_i(t) + \epsilon(t)\mathrm{e}^{\left(\frac{-f(i)}{\lambda^2(t)}\right)} \cdot (\boldsymbol{x} - \boldsymbol{c}_i(t))$$

3. Repeat until a given stopping criteria has been observed.

[1] Many versions of this algorithm choose $\boldsymbol{x}$ using the density of X- however, this is something that is seldom known *a priori*, and in fact, clustering algorithms are used to *estimate* the density!

The function f appearing in Step 2(b) is where the algorithms differ. We will go through the derivation of f in the sections below.

The parameters ϵ and λ are annealing functions of the form:

$$\alpha(t+1) = \alpha_i \left(\frac{\alpha_f}{\alpha_i}\right)^{\frac{t}{T}} \tag{2.1}$$

where α_i, α_f are the initial and final values of the parameter, and T is the maximum number of time iterations [56, 58, 57].

The result of the algorithm produces a clustering that is not unique. Placement depends on the initial conditions and how fast the training parameters ϵ and λ tend to 0. In performing this clustering, one often must make several "passes" at the data, where we use the past ending condition for the new initial condition. This way we can get a better approximation than with a single pass.

Also, since we are choosing data points at random, by using a large number of iterates, we will be shadowing the density of the data. Thus, these algorithms are sensitive to the density and are widely used to approximate it.

Clustering algorithms of this type are Kohonen's SOM and Neural Gas, which are discussed in detail below.

The second class of clustering methods mimic closely what we meant above by a *local* one dimensional data reduction.

Algorithm 2.2 *Clustering by Mean Computation: Type II Algorithm.*

The algorithm is the basis for the Linde-Buzo-Gray (LBG) algorithm [31, 52, 28], and the online version is also known as the k-means clustering algorithm.

1. Initialize the algorithm by randomly placing the centers in $\mathbb{R}^N$.

2. Main Loop:

(a) Sort the data into sets C_i, so that

$$\boldsymbol{x} \in C_i \Leftrightarrow d(\boldsymbol{x}, \boldsymbol{c}_i) \leq d(\boldsymbol{x}, \boldsymbol{c}_j), \quad \forall\, i \neq j$$

Let $i(k), k = 1, 2, \ldots, p_i$ denote the indices for the data $\boldsymbol{x}$ chosen to be in cluster i, so that cluster i has p_i points.

(b) Reset $\boldsymbol{c}_i$ so that it is the mean of the data set C_i.

$$\boldsymbol{c}_i = \frac{1}{p_i} \sum_{k=1}^{p_i} \boldsymbol{x}^{i(k)}$$

(c) Compute the distortion measure on each cluster:

$$M_i = \frac{1}{p_i} \sum_{k=1}^{p_i} d(\boldsymbol{x}^{(k)}, \boldsymbol{c}_{m(k)})$$

(d) Repeat until the distortion measure is below a preset threshold.

For the LBG algorithm, set the distance function to the Euclidean metric:

$$d(\boldsymbol{x}^{(k)}, \boldsymbol{c}_{m(k)}) = \|\boldsymbol{x}^{(k)} - \boldsymbol{c}_{m(k)}\|_2$$

The algorithms presented of this type, the VQPCA and Space Time LBG, differ in the representation of d. This should be chosen so that the metric reflects that property of the data by which the quality of the clustering will be measured. We discuss the objective functions defining the quality in the sections that follow.

The algorithms presented below represent the state of the art in data clustering. We classify them according to template types, discuss each in detail, then we summarize them with a table. The shortcomings we see in the first of the methods is overcome by the Space Time LBG algorithm which is discussed last, and proves to be the optimal technique for the neural charts algorithm.

2.2 Kohonen's Map, or SOM

The Self-Organizing Map (SOM) algorithm is a clustering technique of the first type which is based on a hypothetical self-organizing model of the visual cortex [51]. The self-organizing principle is that the cluster centers are not allowed to freely wander through the data. It is supposed that there exists, in an auxillary space, relationships between cluster centers that are not present in the data space. The primary purpose of this auxillary space is to set up a local ordering of the centers. By that we mean that, in the auxillary space, we define a new metric by which we can measure the closeness of cluster centers.

The key idea here is that each cluster center has *two* representations: A representation in the data set $X \subset \mathbb{R}^N$, and a representation in an auxillary set $\mathcal{I} \subset \mathbb{R}^K$. There need be no relationship between K and N.

The end result of Kohonen's Map will be to transport the ordering defined via $\mathcal{I}$ into the data set X.

Algorithm 2.3 *Kohonen's Map*

1. Construct the index set $\mathcal{I}$ and metric, $d_{\mathcal{I}}$. The index set is constructed in such a way as to make the metric the standard Euclidean metric. The metric uses the indices of the centers as its arguments.

2. Construct $f(i)$ for Algorithm Type I (Soft Classifier):

   (a) Find the center whose Euclidean distance from $\mathbf{x}$ is smallest,

   $$\mathbf{c}_w \in \{\mathbf{c}_i\}_{i \in \mathcal{I}}$$

   This center is referred to as the winner.

   (b) $f(i) \equiv d_{\mathcal{I}}(i, w)$

3. Then the update is given by:

$$\boldsymbol{c}_i(t+1) = \boldsymbol{c}_i(t) + \epsilon(t)\mathrm{e}^{\left(\frac{-f(i)}{\lambda^2(t)}\right)} \cdot (\boldsymbol{x} - \boldsymbol{c}_i(t))$$

Large values of λ are initially used (i.e., the range of influence of the exponential should be about $\frac{1}{2}\mathrm{diam}(X)$), and this strongly enforces the ordering imposed by the index set metric. As the value of λ decreases, the enforcement of the ordering relaxes and the cluster centers are allowed to approximate the point distribution. This technique is illustrated in the example of the torus in Chapter 6.

There is not a known potential energy that the SOM algorithm will minimize. For its original purposes, this was not the goal. Rather, the goal of the original algorithm was to obtain a "self-organization" of the cluster centers.

In this sense, we have some analytical results that the algorithm converges with respect to the ordering. Kohonen proves the following in his book ([51]), for the case that the data and the index set is one dimensional:

Proposition 2.1 *The centers become ordered with probability one, as $t \to \infty$.*

And, as a corollary:

Corollary 2.1 *If the centers are ordered, then they cannot become unordered in future updating.*

A proof for the general case does not yet exist, although Kohonen does give some heuristic arguments.

In the literature, many researchers utilize a two-dimensional lattice for the index space (see, for example [51] and the references contained therein). However, this is not necessitated by the algorithm, and in a previous paper [38] (summarized in Chapter 4) we've shown the utility of using a spherical index set for creating a triangulation of data representing the lobes of the lung. In an example in Chapter 6, the knowledge that the attractor is a torus has been exploited in that the index set has been constructed to reflect that topology (see Figure 6.1).

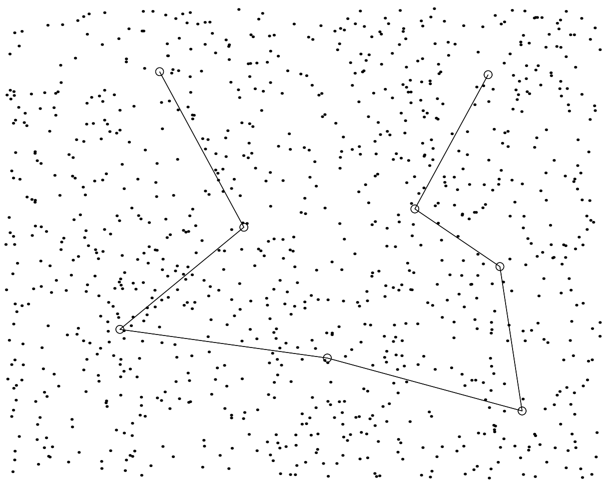

Figure 2.2: Kohonen's Map using 8 cluster centers and a linear index set. The connections are shown in the original data.

Examples of Clustering with the SOM.

In this set of "toy" problems, we use a data set that is randomly distributed across the unit square.

In the first example (Figure (2.2)), the index set was one dimensional: In the auxillary space, $\mathcal{I}$, the 8 centers have the representation: $\{\frac{1}{8}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}, 1.0\}$. The resulting picture is an attempt to map the line to the square. The mismatch in dimension has produced a mapping that takes two point far away in the linear metric, and must map them close together.

In the second example (Figure (2.3)), the index set was two dimensional and had 9 centers, summarized in the 2 by 9 matrix below:

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.5 & 0.5 & 0.5 & 1.0 & 1.0 & 1.0 \\ 0.0 & 0.5 & 1.0 & 0.0 & 0.5 & 1.0 & 0.0 & 0.5 & 1.0 \end{pmatrix}$$

In this case, the topology of the data set matches the topology of the index set.

In the third example (Figure (2.4)), we place 289 clusters arranged on the surface of a sphere, and map these clusters to the unit square. Since we know that $\mathbb{R}^2$ and $\mathbb{R}^3$ are not topologically equivalent, we know that the ordering imposed

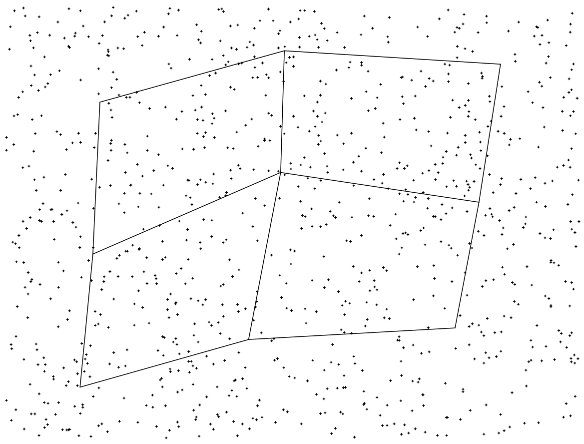

Figure 2.3: Kohonen's Map using 9 centers with a two dimensional lattice ordering. This mapping is topology preserving.

on the surface of the sphere must be violated after mapping it to the square. We will come back to this point in Section 2.6.

These examples highlight the problems and the attributes of Kohonen's Map. On the one hand, if we need to preserve a prior structure on the cluster centers then this algorithm will give the desired result. For example, the biological motivation was for the visual cortex, where the "cluster centers" are arranged on a plane. On the other hand, if one is trying to uncover the topological details of the data, then this algorithm will not suffice. This is the motivation behind the next algorithm, the Neural Gas algorithm.

2.3 The Neural Gas Algorithm

The Neural Gas algorithm ([56, 58, 57, 23]) does away with the prior estimate of the topology, but still maintains an auxillary metric by which to keep the cluster centers inside a competing force field.

The key element in the SOM of the previous section was the ordering of the index set $\mathcal{I}$. The Neural Gas algorithm modulates the update of the force on each cluster center *by its current order in space*. That is, if we have currently chosen a

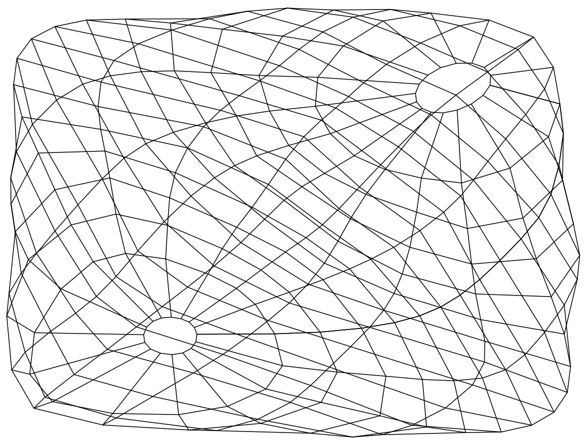

Figure 2.4: Kohonen's Map using a three-dimensional, spherical lattice ordering on the centers. The data set is the same as in Figure 2.3. This mapping is not topology preserving.

data point $\boldsymbol{x}$, then the cluster center closest to $\boldsymbol{x}$ is attracted using the strongest force. The cluster center farthest from $\boldsymbol{x}$ is attracted the least. One may consider this an auxillary metric $\mathcal{I}$, where $\mathcal{I} = \mathbb{R}^N$. Numerically, this can be a problem with a large number of cluster centers, since we will have to sort the distances from $\boldsymbol{x}$ to every cluster center at every iteration.

The algorithm proceeds as follows:

Algorithm 2.4 *The Neural Gas Algorithm*

Follow the Type I algorithm, with the following changes to compute $f(i)$:

1. Sort the distances from $\{\mathbf{c}_i\}_{i=1}^{p}$ to the chosen $\mathbf{x}$.

2. Set the value of $f(i) \equiv s_i$, where s_i is the number of cluster points closer to $\mathbf{x}$ than $\mathbf{c}_i$. For example, if $\mathbf{c}_2$ is closest to $\mathbf{x}$, then $s_2 = 0$, and if $\mathbf{c}_5$ were next closest, then $s_5 = 1$, etc. If the number of cluster points is too high to perform a full sort, one can modify this step to only sort the first m closest cluster points.

The Neural Gas algorithm is designed to minimize an energy potential (which is the distortion error, or reconstruction error):

$$\int \|\boldsymbol{x} - \boldsymbol{c}_i\|^2 P(\boldsymbol{x})\, d\boldsymbol{x}$$

where P is the density on X. and it is shown in [58] that the Neural Gas algorithm is a solution to the minimization problem via a stochastic descent using the gradient. In practice, we are minimizing the distortion error of the LBG:

$$\frac{1}{P}\sum_{i=1}^{P} \|\boldsymbol{x}^{(i)} - \boldsymbol{c}_{m(i)}\|^2$$

which leads back to the local one dimensional reduction.

It is also stated in [58] that the straightforward method to minimize this quantity is the LBG algorithm but that this yields suboptimal placement of the centers. We note that in the Space Time LBG algorithm that will be discussed later, there is a tree structure mitigating the suboptimal placement. That is, the tree will subdivide until a preset minimal distortion error is achieved.

Other versions of the Neural Gas have been proposed, such as the Growing Neural Gas [23], which will determine the number of clusters necessary to reduce the distortion measure to a preset level. This is also a global algorithm based on the spatial distortion measure of the standard algorithm.

We now proceed with a simple example of the Neural Gas algorithm to compare its output with the other algorithms.

Examples using the Neural Gas Algorithm

To compare the Neural Gas algorithm to Kohonen's SOM, we have kept the data set to be a uniform distribution of points in the unit square. Figure 2.5 shows the result of using the Neural Gas algorithm. The connections are drawn from the connectivity matrix C (discussed further in Section 2.6) from the algorithm.

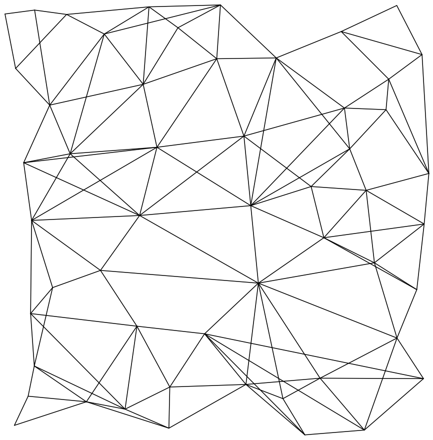

Figure 2.5: The Neural Gas algorithm on the Unit Square data

Here we see a partial triangulation of the data — a triangulation that preserves the topology (again, we defer until later a discussion of the topological aspects of the clustering).

In summary, the Neural Gas algorithm is designed to find an optimal solution to the global (versus tree) cluster placement that minimizes the distortion error given in the LBG algorithm.

2.4 VQPCA

In [45] a clustering scheme is presented that is a hybrid of a general vector quantization routine with local KL (or PCA). The KL transformation is discussed in detail in the next chapter. The result of this hybrid is a modification of the LBG algorithm [52] that uses the local KL to determine the goodness of the clustering. Generally, once a clustering has been obtained, then local KL is performed on the cluster, producing a local basis for the cluster. The first m basis vectors are retained, the data is projected onto them, and an error is computed that represents the distance between the projection of the data and the original data.

More precisely, if A_i denotes the matrix representation of the projection, then

$$A^T_{m(\mu)} A_{m(\mu)} (\boldsymbol{x}^{(\mu)} - \boldsymbol{c}_{m(\mu)})$$

denotes the linear reconstruction of $\boldsymbol{x}^{(\mu)} - \boldsymbol{c}_{m(\mu)}$. Let the error for the data point indexed by μ be denoted by:

$$d_{\mathrm{vq}}(\boldsymbol{x}^{(\mu)}, \boldsymbol{c}_{m(\mu)}) = \|\boldsymbol{x}^{(\mu)} - \boldsymbol{c}_{m(\mu)} - A^T_{m(\mu)} A_{m(\mu)} (\boldsymbol{x}^{(\mu)} - \boldsymbol{c}_{m(\mu)})\|_2 \tag{2.2}$$

Thus, the error measure on cluster i by the linear projection is:

$$\frac{1}{p_i} \sum_{\mu=1}^{p_i} d_{\mathrm{vq}}(\boldsymbol{x}^{(\mu)}, \boldsymbol{c}_{m(\mu)}) \tag{2.3}$$

where p_i denotes the number of points in cluster i. This algorithm is directly designed to minimize the error after locally projecting the data onto its m dimensional KL basis. Thus, this algorithm is designed to find the best cluster placement that yields the best local $m-$dimensional representation.

The algorithm proceeds as follows:

Algorithm 2.5 *VQPCA Clustering Algorithm*

Use Algorithm Type II to train, with the following specifications:

Before starting the algorithm, initialize the value of m, called the local target dimension. This initialization is ad-hoc, and represents a guess as to what the general local dimension is.

For each iteration of the main loop:

1. Perform local KL within each cluster, retaining m eigenvectors.

2. Project the data onto the m-dimensional hyperplane defined by the eigenvectors above.

3. Set $d \equiv d_{\mathrm{vq}}$, defined in Equation (2.2).

Numerical results in [45] suggests that more accurate reconstructions are obtained using this procedure versus compression using a global bottleneck neural network (discussed in Chapter 4) or using a standard vector quantization routine.

2.5 Space Time LBG Clustering

This section specifically addresses the role of data clustering in the context of preparing the data set for a subsequent local modelling of the dynamical system from whence the data came. That is, the data set X is representative of data obtained via some dynamical system, and lies on some invariant, attractive manifold. The task is then to separate the data set X so that each cluster represents a homogeneous dynamical regime.

By this we will mean that each cluster represents a cohesive spatial region on the manifold, and the time derivative of the flow in the cluster is almost constant.

We note that these two conditions are not necessarily independent. Indeed, if the spatial clusters are small enough, and the dynamical system is smooth, then the second condition holds automatically. Furthermore, if we induce the trivial clustering — each data point represents a unique cluster — then both conditions hold.

On the other hand, if we separate the data into p clusters (where p is some small number) using only the spatial data, then the second condition will generally fail. If we separate the data using only the derivative information, then the clusters will generally be spatially far apart.

What makes the problem non-trivial is to use the smallest number of clusters possible so that both conditions hold. From the notes above, we see that the problem has a solution as the number of clusters reaches the number of data points.

In terms of locally modeling a dynamical system, local clustering has been done in the context of restricting the flow to a given compact manifold ([11]). In this case, the manifold is known analytically, and the authors suggest clustering the manifold as ϵ-sized balls. This will work well if the manifold is well-behaved

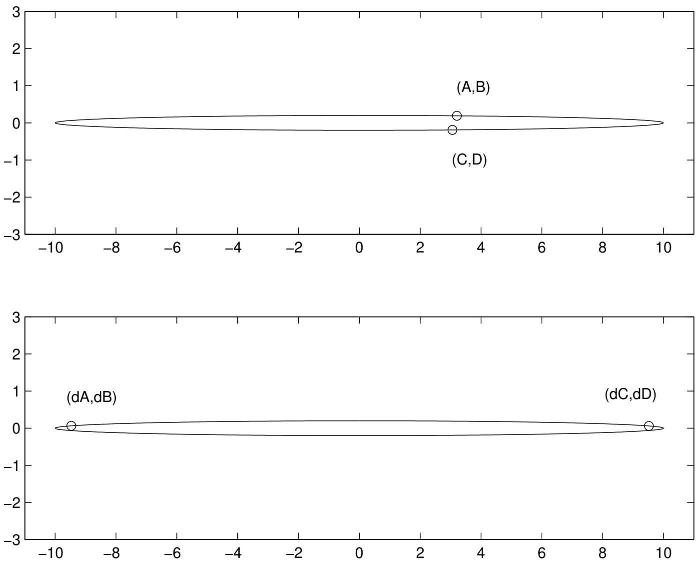

Figure 2.6: The thin ellipse. Upper picture is the original data set, and lower picture are the derivatives. Points (A,B) and (C,D) are spatially very close, but temporally far away.

(i.e., the graph of an analytic function), but we would like our clustering algorithm to be adaptive — we should use more clusters in regions where the flow changes rapidly, less clusters where the flow is almost constant.

The Space Time LBG deals with two of the difficulties of clustering spatial data in preparation for constructing local models: adaptivity as discussed in the previous paragraph, and resolution of class information. Spatially, there may be confusion as to class membership, i.e., two data points may be very close while the derivative information is indicating that the two points should belong in different dynamical regimes.

Consider the following example, where we suppose that the flow is given by:

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} 10\cos(t) \\ \frac{1}{5}\sin(t) \end{pmatrix}$$

which constructs a manifold which is long and very thin, as depicted in Figure 2.6.

Now suppose that this manifold is discretely sampled, and we are attempting to cluster it. Then using a clustering algorithm based solely on spatial information, points such as (A, B) and (C, D) will almost surely be classified in the same cluster, while they belong to very different temporal classes. The temporal derivatives of x and y are plotted in the second image, together with the derivatives of our two sample points, (dA, dB) and (dC, dD). Here it is clear that they do not belong in the same class — with the idea that we wish points in the same cluster to have the same qualitative dynamics. The question now is how to incorporate the temporal information into the clustering.

The key concept is that we think of the data set as being the end result of a bad projection — points that should be classified as being from distinct regimes are projected together. By an augmentation of the domain of the decision function, we hope to clarify this conflicting information. In other words, we increase the dimensionality of the data so that, for each $\boldsymbol{x}$ in a given class, there exists an ϵ so that, if $\bar{\boldsymbol{x}}$ is another point in the ϵ neighborhood, then

$$d(\boldsymbol{x}, \bar{\boldsymbol{x}}) < \epsilon \Rightarrow \bar{\boldsymbol{x}} \text{ is of the same class as } \boldsymbol{x}$$

In the case of data derived from a dynamical system, each $\boldsymbol{x}$ has an associated time derivate, $\dot{\boldsymbol{x}}$, which may be analytic or numerically estimated. The definition of what we mean by a good clustering then naturally leads to the augmented data set:

$$\hat{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix}$$

with the modified Euclidean metric:

$$d_{\mathrm{ST}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \|A(\hat{\boldsymbol{x}} - \hat{\boldsymbol{y}})\|_2$$

where the matrix A is a positive, diagonal matrix. The role of A is to provide a scaling of the data so that the augmented information is comparable to the original data. We will see the effect of A in the examples below.

The augmented domain and metric is then presented to the existing clustering algorithm for processing. In the examples that follow, the basic clustering algorithm is the LBG algorithm, although space-time can be incorporated into the

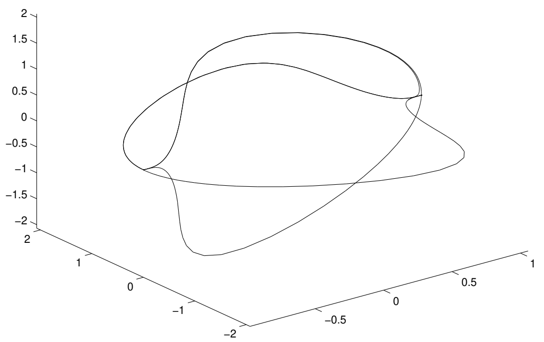

Figure 2.7: An attracting set with an extreme point distribution. Approximately 90% of the data is within a small ball about the two sharp corners. Let A be the corner on the left, and B be the opposing corner.

other algorithms as well. In Section 2.3 on the Neural Gas algorithm, it was stated that the LBG algorithm may yield suboptimal placement of the cluster centers, and we stated that this was mitigated by the tree structure that we will use. There is another problem associated with data sets from dynamical systems that make the Neural Gas algorithm not appropriate for our general use.

If we could choose the distribution of points on the manifold, we would choose a uniform distribution, so that all areas of the manifold are represented equally well. This is not the case for some attracting sets. We consider an extreme example in Figure 2.7. Points A and B represent unstable fixed points for the attracting set. The velocity of the trajectory is relatively very fast until it approaches points A or B, where time scale slows down considerably. Taking a sampling at equal time intervals means that the distribution of the data is heavily concentrated at A and B. Since Neural Gas is sensitive to the point distribution, it would result in most of the cluster centers being concentrated at the points A and B, leaving the rest of the manifold with little (if any) representation in terms of the cluster centers.

With the LBG algorithm, we have much less sensitivity to the point distribution, and will therefore have a much better representation of the full attracting set. We have chosen the LBG with this example in mind.

Heuristically, one can visualize the approximating manifold for the attracting set as being locally constructed in such a way so that the temporal derivatives are parallel. Thus, we would like to incorporate this heuristic into the clustering algorithm. We note that we are not interested in the speed of the flow (the size of the temporal derivative), just in the direction. In what follows, we therefore assume that the temporal derivative vectors $\{\dot{\boldsymbol{x}}_i\}_{i=1}^{|X|}$ ($|\cdot|$ is the cardinality of $\cdot$) have been normalized. In what follows, take C_i to represent the i^{th} cluster, and $\boldsymbol{c}_i$ be the i^{th} cluster center. Furthermore, let

$$d_{\text{ST}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{c}}_i) = \alpha \|\boldsymbol{x} - \boldsymbol{c}_i\| + \beta \|\dot{\boldsymbol{x}} - \gamma_i \dot{\boldsymbol{c}}_i\|$$

where α, β are the diagonal elements of A that are problem dependent, and $\gamma_i = \frac{1}{\|\dot{\boldsymbol{c}}_i\|}$. With the extra condition that $\alpha + \beta = 1$, we can substitute $\beta = 1 - \alpha$, and thus have only one extra parameter. Also, we note here that $\dot{\boldsymbol{c}}$ will denote a cluster center for the time derivatives of X, and not $\frac{d}{dt}(\boldsymbol{c})$.

Algorithm 2.6 *Space Time LBG Clustering*

The algorithm proceeds as follows. We include all steps for clarity.

1. Sort the data so that $\boldsymbol{x} \in C_i$ iff:

$$d_{\text{ST}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{c}}_i) \leq d_{\text{ST}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{c}}_j) \; \forall i \neq j$$

2. Reset the centers:

$$\boldsymbol{c}_i(t+1) = \frac{1}{|C_i|} \sum_{k=1}^{|C_i|} \boldsymbol{x}_k$$

$$\dot{\boldsymbol{c}}_i(t+1) = \frac{1}{|C_i|} \sum_{k=1}^{|C_i|} \dot{\boldsymbol{x}}_k$$

where $\boldsymbol{x}_k \in C_i$, and $i = 1, 2, \ldots, P$.

3. Repeat until the distortion,

$$D = \frac{1}{|X|} \sum_{k=1}^{|X|} d_{\mathrm{ST}}(\hat{\boldsymbol{x}}^k, \hat{\boldsymbol{c}}_{m(k)})$$

is no longer decreasing.

The point of the clustering is to seperate the attracting set into dynamically distinct regimes. In particular, we do not want the flow direction at the outer points of the cluster to become orthogonal to the flow direction at the center. We incorporate this by requiring that, for each cluster, the maximum distance between any flow direction $\dot{\boldsymbol{x}}$ and its center $\dot{\boldsymbol{c}}_{m(x)}$ be less than a prescribed ϵ_1:

$$\max_i \|\dot{\boldsymbol{x}}^{(i)} - \gamma_{m(i)} \dot{\boldsymbol{c}}_{m(i)}\| \leq \epsilon_1$$

Thus, clustering will proceed until this criterion has been attained.

The LBG algorithm, and its modification outlined above, tend to cluster a data set into regions of approximately equal size (depending on the value of α, and if the distribution of the data is approximately uniform). We would like our algorithm to be capable of targeting those regions of the attracting set that exhibit a high degree spatio-temporal complexity. Therefore, this algorithm is implemented by using a binary tree clustering algorithm.

The root of the tree is initialized as the global centroid of the data, with the derivative similarly initialized. Subsequent branching of the tree is initialized as a randomly small perturbation of the center, so that the two spawned centers, $\tilde{\boldsymbol{c}}_{i,j}$ are initialized from the parent node, $\boldsymbol{c}_k$ as:

$$\tilde{\boldsymbol{c}}_{i,j} = \boldsymbol{c}_k \pm \epsilon_2$$

The Space Time LBG algorithm is applied to only two clusters at a time, with subsequent branching of the tree until the stopping criterion has been attained for each cluster. The details of the tree algorithm are discussed in Chapter 5.

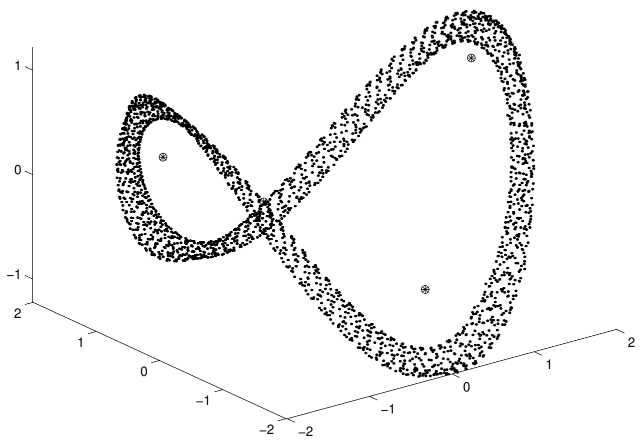

Figure 2.8: Spatial clustering using four cluster centers.

Examples using the Space Time LBG

The data set we are working with in this example is similar in spirit to the ellipse given above. It is a thin torus, which will be revisited in Chapter 6. The clustering of this set is presented at this time in order to understand the motivation for the Space Time LBG.

The spatial algorithms from previous sections will tend to cluster the torus as if it were a cycle. In fact, in order to uncover the topology of the torus, one would require enough cluster points so that the distance between cluster centers could be smaller than the inner diameter of the torus. In Figure 2.8, we see the result of having too few clusters. In Figure 2.9 we can see immediately that this number would have to be very large indeed.

In the algorithms that we implement, the matrix A has the form

$$A = \mathrm{diag}\{\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n\}$$

where the α and β coefficients serve to scale the metric so that the data and the derivatives are comparable (often the derivative time scale is orders of magnitude larger than the data scale). In the first two figures of the torus, β was set to 0, and

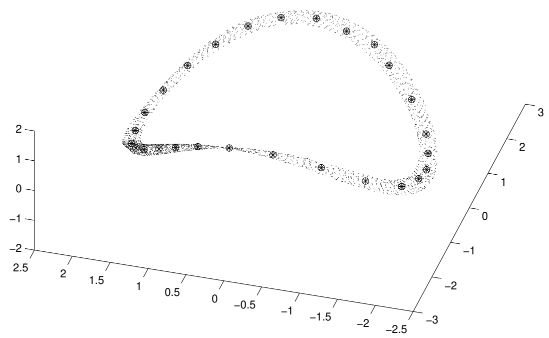

Figure 2.9: With $\beta = 0$, the standard LBG algorithm does not recognize the toral shape without using a large number of clusters.

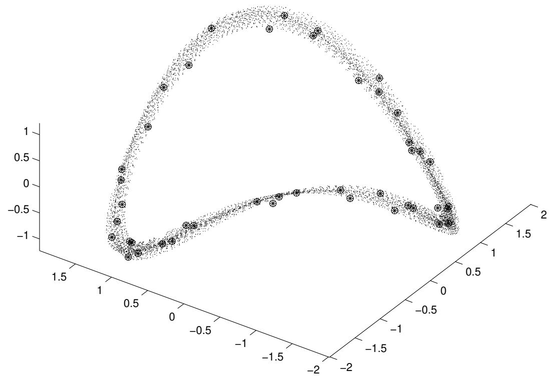

Figure 2.10: Space Time LBG clustering of the Thin (20 dimensional) Torus

α was 1. In the third, the data and the derivatives were scaled in preprocessing, and the values of α and β were 0.5.

In Figure 2.9, we see the typical result of either Kohonen's Map, Neural Gas, or the LBG algorithm. The clustering has misinterpreted the torus. On the other hand, in Figure 2.10, we see that the cluster centers coming out of the Space Time LBG algorithm have placed themselves on the exterior of the torus showing how we have been able to incorporate the requirement that clusters be "dynamically homogenous regimes".

2.6 Constructing Topological Features

As mentioned above, the SOM forces a prior estimate on the structure of the manifold. If the topology of the index space is not equivalent to the topology of the manifold, then one cannot form a map to the manifold that is *topology preserving*, i.e., the clustering should map neighboring points in the index metric to neighboring centers on the manifold, and should map neighboring points on the manifold to neighboring center indices (see, for example, [56])).

We see in Figures 2.2 and 2.4 that the topologies of the index sets do not match with the topology of the data set. On the other hand, the topological connections shown in Figures 2.3 and 2.5 match the topology well. In [57], the ideas of topology preserving maps are presented in the context of Delauney Triangulation. These ideas culminate in an algorithm that can, given a placement of cluster centers, inscribe edges that give a partial triangulation of the data that is consistent with its topology. In [57] this algorithm is incorporated into the Neural Gas algorithm directly, but we treat it separately since it can be implemented *after* any of the clustering algorithms presented with the exception of Kohonens SOM. In doing so, we call this the Topology Constructing Algorithm (TCA).

Rather than having the index space as in Kohonen's SOM, the TCA constructs a "connectivity matrix" C, with:

$$C_{ij} = \begin{cases} 0 & \text{if } \mathbf{c}_i \text{ is not joined to } \mathbf{c}_j \\ 1 & \text{if } \mathbf{c}_i \text{ is joined to } \mathbf{c}_j \end{cases}$$

The algorithm begins with C initialized to the zero matrix, and as the centers are updated, if $\mathbf{c}_a$ is the closest center to the data point $\mathbf{x}$, and $\mathbf{c}_b$ is the next closest, then C_{ab} is set to 1. The age of the connection must also be tracked, removing it if the connection is not regularly refreshed.

Algorithm 2.7 *Topology Constructing Algorithm (TCA)*

1. Allocate space for the connectivity matrix C and the time matrix T, so that each has dimension $p \times p$, where p is the number of cluster centers. Initialize both to zero. Set an annealing parameter T, which will control the maximum age allowed for the connections.

2. Main Loop:

   (a) Choose $\boldsymbol{x}$ at random, and find its two closest centers.

   (b) Update the Connection and Time matrices: If $\mathbf{c}_a$ is the closest center, and $\mathbf{c}_b$ is next closest, then set $C_{ab} = 1$, and set $T_{ab} = 0$. Adjust all time entries by adding 1 to each T_{ij}. If $T_{ij} > \mathrm{T}_{\text{current}}$, set both C_{ij} and T_{ij} to zero.

We view the TCA as constructing a topology preserving map in the following sense. If the edge $C_{ab} = 1$, then the cluster indices a and b are neighbors in the index space. By definition, nearest neighbors in the index space are then identified with nearest cluster centers in data space, and nearest cluster centers in data space are identified with nearest neighbors in index space.

With this algorithm we can uncover topological features of the data by first using any of the clustering algorithms, with the exception of the SOM, since the topology is defined prior to clustering.

	Type	d or f	Purpose or Measure	Computational Concerns	Best for:
SOM	1	$d_{\mathcal{I}}(i, w)$	Transport of Top. Structure	$\|\|x - c_i\|\|$ at each step. d at each update. Needs several passes.	Known Structure
Neural Gas	1	s_i	$\min \frac{1}{\|X\|} \sum \|\|\hat{x}\|\|$	$\|X\|$ norm comps, then a p point sort. Needs several passes	Optimal for global clustering
VQPCA	2	$\|\|\hat{x} - A_i^T A_i \hat{x}\|\|$	Minimize d.	$\|X\|p$ norms at each step and local SVD	Alternative to Bottleneck.
LBG	2	$\|\|\hat{x}\|\|$	Minimize d.	$\|X\|p$ norms	With a tree, a quicker alternative to Neural Gas
Space Time	2	$\alpha\|\|\hat{x}\|\| + \beta\|\|\hat{x}'\|\|$	Minimize d.	$\|X\|p$ norms	Attracting sets of dynamical systems.

Table 2.1: A Comparison of Clustering Methods. $\hat{x} = \boldsymbol{x} - \boldsymbol{c}_i$

2.7 Discussion and Summary

The result of the clustering algorithm is a set of subsets of the manifold, $\{C_j\}_{j=1}^p$, with corresponding cluster centers $\{\mathbf{c}_j\}_{j=1}^p$. The "partitioning" criterion also yields a decision function which are summarized in Table 2.1.

In practice, when the decision function is applied, one rarely encounters a tie, and in that case, since in our training of the inverse map, each of the C_i were slightly expanded, it becomes inconsequential as to which of the competing domains $\mathbf{x}$ is assigned (see Chapter 5 for more details). The clustering algorithms are summarized and compared in Table 2.1.

The neural clustering algorithms are extremely sensitive to the point distribution on the manifold (in the algorithm, this is the line "Take $\boldsymbol{x}$ at random."). In fact, this property has been exploited as a basis for constructing probability estimates in forecasting time series (see, for example [58] and [18]).

The ideal clustering algorithm will provide a spatio-temporal clustering with the property that each cluster is homogenous in space and time. In particular, it will group elements of the flow in such a way as to distinguish between dynamical regimes. The spatial based clustering lacks an encoding of temporal information, and this is adequate when the cluster consists of trajectories that are very similar. However, for more complex flows this may not be the case. Numerical experiments suggest that some chaotic flows (i.e., the Kuramoto-Sivashinsky Equation, $\alpha = 68.0$) consists of multiple low dimensional semi-stable states that are connected by bursting behavior [40, 41]. It is desireable to separate distinct dynamical events that exist spatially close, but are temporally very distinct. The resulting groups are what we refer to as dynamical regimes, and are the result of a hierarchical Space Time LBG clustering method. The first stage yields a large, space-time covering and this is followed by a subgrouping in each cluster. If the grouping does not allow for an accurate reconstruction, then the cluster is subdivided, and the routine continues until clusters are obtained that are spatially and temporally homogenous.

There are advantages and disadvantages associated with each of the algorithms. The main consideration in choosing is based on a familiarity with the data. For example, if the topological structure is known and we would like to transport a given ordering into the data, then the SOM is the algorithm of choice. If the data is distributed uniformly, and we want a global (versus tree) clustering, then the Neural Gas algorithm is designed to give the best results. If local, m dimensional representations are desired, then we use the VQPCA. For attracting sets of dynamical systems, we have found the Space Time LBG with the tree clustering to give the best results, since it has been designed with this task in mind.

For the neural charts algorithm, the centers $\{\mathbf{c}_i\}_{i=1}^{p}$ obtained by the clustering algorithm are used to represent the local origins for the local tangent spaces. In the next chapter, we build the local tangent spaces.

Chapter 3

KARHUNEN-LOÈVE (KL) EXPANSIONS

In this chapter, we review the Karhunen-Loève expansion and the methods derived from it that determine optimal coordinate systems and local dimensionality. From these, we construct the Modified Local SVD algorithm which is designed to automate this determination and in the context of the neural charts algorithm, is used to obtain the number of basis vectors for the local coordinate systems.

3.1 Introduction

The key step in modeling is to determine the smallest number of degrees of freedom in the model. Suppose we are modeling a dynamical system in $\mathbb{R}^n$ and the long term dynamics of the system actually lie on a manifold of much smaller dimension, k. Then locally, we should be able to describe the dynamics using k equations of motion, versus the original n. In order to obtain these local descriptions, it is required that we obtain a local description of the manifold. By this we mean that the manifold has been decomposed into subsets that are adequately small enough so that the subset is homeomorphic to $\mathbb{R}^k$ for some value of k.

The appropriate clustering routines by which we obtain the cluster centers are discussed in Chapter 2. For now, let it suffice that the data set has been separated into ϵ-balls, with centers given by $\{c_i\}_{i=1}^{p}$. Formally, we think of these centers as points on a manifold at which we will compute the Taylor expansion of the solution to the dynamical system which has been sampled at some finite set of discrete times, so that:

$$X(t) = (x_1(t), x_2(t), \ldots, x_n(t))^{\mathrm{T}} \quad t \in \{t_1, \ldots, t_P\}$$

We also assume that the centers, $\{c_i\}_{i=1}^{p}$ lie on the manifold, so that at some t_i^*, $X(t_i^*) = c_i$. Then the local coordinate system we use is the span of the Jacobian of the manifold at c_i, and using this k-dimensional basis, we can identify each point of the dynamical system with a point on the manifold,

$$X(t) \rightarrow (y_1, y_2, \ldots, y_k, f_1(y_1, \ldots, y_k), \ldots, f_{n-k}(y_1, \ldots, y_k)) \tag{3.1}$$

where the f_i are smooth scalar valued functions. There is no guarantee that the first order Taylor terms will be dominant. We discuss the connection in more detail in Section 3.4 and Appendix C.

Given this coordinate system, the dimensionality reduction is performed by the projection of X onto the coordinate system:

$$\pi_i(X(t)) = (y_1, \ldots, y_k)$$

It will be the task of the neural networks, discussed in Chapter 5, to build the inverse of π_i, so we leave this aspect for now.

The problem then is to determine the value of k based at each c_i, and to determine the basis vectors we will use.

In general, the long term dynamics may not lie directly on a manifold — in fact the attracting set may be fractal. Also, the data may not lie in a local surface at all, which may relate to regions in phase space where flows conjoin or may exhibit bursting behavior. Isolation of these regions of phase space is one goal of local modeling. In such cases, we are looking for the smallest value of k such that the attracting set can be embedded in a manifold of dimension k. This will mean that the f defined in Equation (3.1) will have the form:

$$X(t) \rightarrow (y_1, y_2, \ldots, y_k, 0, \ldots, 0) \tag{3.2}$$

so that the projection encapsulates the data.

The algorithm that determines dimensionality should lend itself to automation. Although some user control will probably be necessary (and sometimes desireable), there should exist defaults by which the algorithm can proceed with minimal expert input.

This is the drawback to the current algorithms, which are described below. The key tool is the Karhunen-Loève (KL) expansion (which is also described in the literature as Singular Value Decomposition (SVD), or Principle Component Analysis (PCA), or Proper Orthogonal Decomposition (POD)). These algorithms have been used for many years and are well known for their global compression and noise reduction abilities. Less used are algorithms for local representations, and this is probably due to the problem of determining suitable local regions, and in automating the process.

An overestimation of the dimension will still yield good results for the subsequent projection and neural approximation, while underestimation will result in an inability of the neural networks to reconstruct the data accurately. Thus, while a true estimation of the dimension of the cluster would be worthwhile, for the neural charts algorithm it will suffice to determine a dimensionality small enough to be an approximation of the true dimension, while large enough to allow for an accurate reconstruction. This "error catch" is included in the full neural charts algorithm.

The sections are organized as follows. First, we present the Karhunen-Loève expansion, on which all of the methods are built. Fukunaga's method is then presented followed by the local SVD algorithm by Broomhead et. al. and the Modified SVD algorithm, with examples.

3.2 Karhunen-Loève (KL) Expansion

We begin with a brief outline of the KL expansion whose purpose is to provide an optimal orthonormal coordinate system for a given data set X, based at its mean, m.

Let P be the number of points in X. Let $\tilde{X}$ be the matrix formed by taking the data point $\mathbf{x}_i - m \in \boldsymbol{R}^n$ as the i^{th} row of $\tilde{X}$. Thus, $\tilde{X}$ is a matrix that has dimensions P x n. The KL algorithm then proceeds:

Algorithm 3.1 *The Karhunen-Loève Expansion*

1. Compute the ensemble-averaged covariance matrix, $A = \frac{1}{P}\tilde{X}^T\tilde{X}$, or

$$A_{ij} = \frac{1}{P}\sum_{k=0}^{P-1} x_{ki}x_{kj}$$

where x_{kl} is the l^{th} coordinate of the k^{th} data point.

2. Solve for the eigenvectors and eigenvalues of A,

$$A\boldsymbol{\phi}_n = \lambda_n\boldsymbol{\phi}_n$$

Note that A is a positive, symmetric matrix; the eigenvalues are the square of the variances in the direction of the eigenvectors. We assume that the eigenvalues of A are ordered from largest to smallest.

3. Each $\boldsymbol{x}$ can be expressed in the new coordinate system:

$$\boldsymbol{x}^{(\mu)} = \sum_{i=1}^{n} \alpha_i\boldsymbol{\phi}_i$$

where α_i is the inner product between $\boldsymbol{x}^{(\mu)}$ and $\boldsymbol{\phi}_i$.

4. The k-dimensional approximation to $\boldsymbol{x}^{(\mu)}$:

$$\tilde{\boldsymbol{x}}^{(\mu)} = \sum_{i=1}^{k} \alpha_i\boldsymbol{\phi}_i$$

or in matrix notation:

$$\tilde{X} \approx \tilde{X}\Phi(:,1:k)\Phi(:,1:k)^{\mathrm{T}}$$

with truncation error pointwise:

$$\tilde{\tilde{\boldsymbol{x}}}^{(\mu)} = \sum_{i=k+1}^{n} \alpha_i \boldsymbol{\phi}_i$$

or in matrix notation:

$$\tilde{\tilde{X}} = \tilde{X}\Phi(:,k+1:n)\Phi(:,k+1:n)^{\mathrm{T}}$$

The coordinate system chosen by selecting the first k eigenvectors of the covariance matrix will simultaneously optimize several criteria [43]:

- Mean square truncation error.
- Maximal data variance.
- De-correlate the coordinate axes.
- Maximize information compression.

It is remarkable that all of the above are in fact equivalent. The details for the interested reader can be found in [43].

In the context of a linearized flow (our context), the eigenvalues have a special interpretation. Given that

$$\frac{1}{P}X^{\mathrm{T}}X = \Phi\Lambda\Phi^{-1}$$

where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, we can interpret the normalized eigenvalues:

$$\frac{1}{\sum_{i=1}^{n}\lambda_i}\begin{pmatrix}\lambda_1\\ \vdots\\ \lambda_n\end{pmatrix} = \begin{pmatrix}\tilde{\lambda}_1\\ \vdots\\ \tilde{\lambda}_n\end{pmatrix}$$

in terms of the energy of the system $X(t)$. When $X(t)$ represents the Fourier expansion coefficients, for example, then $\tilde{\lambda}_j$ corresponds to the energy in the j^{th} mode, and the partial sum

$$S_j = \sum_{i=1}^{j} \tilde{\lambda}_i$$

represents the energy captured by retaining the first j eigenfunctions. The graph of

$$\{\tilde{\lambda}_j\}_{j=1}^{n} \text{ vs. } j$$

can then be interpreted as the energy distribution across the eigenfunctions.

3.3 Karhunen-Loève Expansion and Dimensionality Estimation

This section is a summary of dimensionality estimation using KL eigenvectors. For a more details, see [25], [24].

This algorithm is a local estimation of dimension based on obtaining "good" clusters of the data. In Fukunaga's work, this clustering is left to an expert to construct and give to the algorithm [25], [24]. Given a local ball of data, $\tilde{X}$, we have:

Algorithm 3.2 *Fukunaga's Dimensionality Estimation Algorithm*

- Compute Φ, Λ so that

$$\frac{1}{P_i} \tilde{X}_i^{\mathrm{T}} \tilde{X}_i = \Phi \Lambda \Phi^{-1}$$

- The dimensionality is determined by finding a k so that, for $j \in \{1, \ldots, k\}$,

$$\tilde{\lambda}_j > \epsilon \tag{3.3}$$

  where ϵ is a dimensionality cutoff that is problem dependent. Thus, there is a range of dimensionalities possible depending on this cutoff value.

Some remarks are in order at this point.

- The dimensionality depends on an ad-hoc cutoff that is problem dependent.
- The maximum dimensionality of a data set is one less than the number of vectors. It is important to insure that there are enough data vectors in each local region to accurately estimate the dimensionality.
- What is the percentage cutoff to use?

3.4 The Local SVD

In this section, the algorithm of Broomhead and colleagues is summarized. For more details, the reader is referred to [12], [11], [19].

Let $\mathcal{M}$ be a k-manifold in $\mathbb{R}^n$, with $k << n$. Then the manifold looks locally like $\mathbb{R}^k$ so that, in a small enough neighborhood of any point $x \in \mathcal{M}$, $\mathcal{M}$ is well approximated by its tangent space at x.

A local analysis of $\mathcal{M}$ at c consists of analyzing an ϵ-neighborhood matrix, $X_\epsilon(c)$, whose rows consist of vectors $(\boldsymbol{x}_j - c)^T$. For sufficiently small ϵ, the rows of $X_\epsilon(c)$ are approximately tangent vectors to $\mathcal{M}$ at c. As $\epsilon \to 0$, the rank of $X_\epsilon(c)$ is the dimension of $\mathcal{M}$ at c.

As ϵ is increased, the first k singular values will scale linearly until saturation or the effects of curvature become noticeable. The remaining singular values will grow as ϵ^2 or faster.

Thus we have the following decomposition:

$$\left(\frac{1}{P}\right) X_\epsilon^{\mathrm{T}} X_\epsilon = V \begin{pmatrix} O(\epsilon^2) I_{k\mathrm{x}k} & 0 \\ 0 & O(\epsilon^4) I_{n-k\mathrm{x}n-k} \end{pmatrix} V^{\mathrm{T}} \tag{3.4}$$

where P is the number of points in X_ϵ, and $O(\epsilon^m) I_{n\mathrm{x}n}$ is a diagonal matrix with entries:

$$c_1\epsilon^m, c_2\epsilon^m, \ldots, c_n\epsilon^m$$

In the local coordinate system given by the columns of V, we can construct the $f(y)$ (from Equation (3.1)) so that $f(0) = 0$ and $Df(0) = 0$. By definition of the k-manifold and how the coordinate system was found, f will have the form:

$$f(y) = y^T D^2 f(0,0) y + O(\|y\|^3)$$

In contrast to Fukunaga's algorithm, this algorithm considers the *scaling* of the singular values, rather than the singular values at a single radius. This is critical in light of the remarks made at the end of the introduction — there is a visual check to determine what terms of the Taylor series are dominant.

We construct the local SVD curves for X_ϵ in the following manner. Let r_i be the distance from $\boldsymbol{x}_i$ to the center c, and $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i - c$.

The local SVD curves are represented by the pairs

$$(r_i, s_i^j), \quad i = 1, 2, \ldots, p$$

where $\{s_i^j\}_{j=1}^n$ are the n singular values for the covariance matrix on the data whose distance to center is less than r_i.

The algorithm for producing the local SVD curves is given below:

Algorithm 3.3 *The Local SVD Algorithm*

- Reorder the data set so that $\tilde{\boldsymbol{x}}_i$ is the i^{th} closest data point to the center.

- Begin the loop by taking the first $n+1$ points of data. Denote by $\tilde{X}(1:i,:)$ the matrix formed by taking the first i center-substracted data points as the i rows.

- Compute the n singular values for

$$\frac{1}{i}\,\tilde{X}(1:i,:)^{\mathrm{T}}\tilde{X}(1:i,:)$$

Since the maximum dimension of the data is one less than the number of points, we must begin with enough data. If we are in $\mathbb{R}^n$, we begin with $n+1$ points. If n is very large with respect to the suspected dimension, k_s, then we require only $k_s + 1$ points to begin the expansion.

The main difficulty in using this algorithm is in the automation process. In order to know the coefficients to the linearly scaled eigenvalues, one needs to know the distribution of the data. Therefore, from one local neighborhood to the next, the scaling coefficients may change radically, depending on the distribution. Seldom is the data uniformly distributed — in fact the data may exhibit a fractal structure.

The images in Figure 3.1 depict the result of using this algorithm applied to a neighborhood on the surface of the unit sphere. It can be shown analytically (see Appendix A) that on the unit sphere with uniform density, the corresponding singular value matrix to Equation (3.4) has the following form (t is radial scaling from center):

$$\begin{pmatrix} \frac{1}{2}t & 0 & 0 \\ 0 & \frac{1}{2}t & 0 \\ 0 & 0 & \frac{1}{\sqrt{12}}t^2 \end{pmatrix} \tag{3.5}$$

The first image shows the numerically determined values, together with the analytic estimates when the data is uniformly distributed. The second image shows the same region in space, but with a non uniform density. If the distribution is not uniform, then, although the two dominant singular values still scale linearly, they have differing slopes. In some cases, this can be a significant difference. Again, this is the main difficulty associated with using this algorithm.

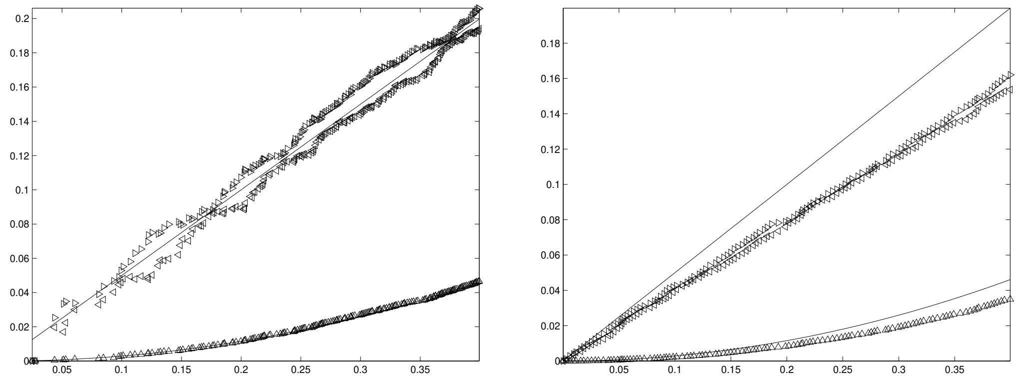

Figure 3.1: SVD Curves for a neigborhood on the Unit Sphere. Numerical values are dotted, analytic estimates are solid. On the left, the spherical cap has a uniform distribution. On the right, the distribution of data was not uniform.

3.5 The Modified Local SVD

We are interested in the dimension of the tangent space, so that for our problem, the best distribution is a uniform distribution of points over the local manifold. This is almost never given, however. Since our data represents the flow of a dynamical system *embedded in* the manifold, the manifold itself will generally not be uniformly represented.

We solve this problem by performing a local *whitening* transformation on the data set [24] This transforms a data set so that its covariance matrix is the identity matrix.

Let the reduced SVD of the P by n matrix X be denoted by

$$X = USV^{\mathrm{T}} \tag{3.6}$$

so that U is a unitary P by k matrix, S is the k by k diagonal matrix of singular values, and V is a unitary k by k matrix. Here we are assuming that the matrix X has rank k. More will be said about the value of k below.

If we were to do a KL decomposition of X, then:

$$\left(\frac{1}{p}\right) X^{\mathrm{T}} X = V \left(\left(\frac{1}{p}\right) S^2\right) V^{\mathrm{T}} \tag{3.7}$$

In matrix notation, the whitening transformation we will use has the form:

$$X \to XV(\sqrt{p})S^{-1} = Y$$

The purpose of the whitening transformation is to change the original hyperellipsoid of local data with differing variances in each eigendirection to a coordinate system where the local data has equal variance in every eigendirection This will standardize the singular value curves, as we will see in a moment.

Proposition 3.1 *The covariance matrix for the locally whitened data is the identity matrix.*

Proof:

$$\frac{1}{p}Y^{\mathrm{T}}Y = \frac{1}{p}\left(\sqrt{p}S^{-1}V^{\mathrm{T}}\ X^{\mathrm{T}}X\ V\sqrt{p}S^{-1}\right)$$

so that

$$\frac{1}{p}Y^{\mathrm{T}}Y = \frac{p}{p}S^{-1}S^{2}S^{-1} = I$$

Remark *In practice, to formulate S^{-1}, we will keep only the k singular values above a noise floor.*

Remark *Full rank (in the case above, rank k) transformations on the manifold will not change its dimension.*

Remark *We use spherical neighborhoods in the whitened data, which correspond to elliptical neighborhoods in the original data. Therefore, one can think of the whitening as a local change of the metric to accommodate the distribution of data.*

Important to the modified algorithm is that, when performing the Local SVD algorithm on the last data point in Y, *all* singular values will be 1. Therefore, if R is the maximum distance from a point in X to the center, and if, at R the singular values all must be unity, then

Proposition 3.2 *The slopes of all linear scaled singular values must be* $\frac{1}{R}$*, and the coefficient for the quadratic coefficient must be* $\frac{1}{R^2}$*.*

Proof: We have that all singular values for the covariance matrix of data on the ball of radius R are 1, and with radius 0 are 0. For radius r_i, we have n singular values, $\{\sigma_i^j\}_{j=1}^n$. Denote the k^{th} SVD curve by the coordinates (r_i, σ_i^k). Then all SVD curves go through the coordinates $(0,0)$ and $(R,1)$. Therefore, the slope of the linear curves must be $\frac{1}{R}$, and the quadratic coefficient must be $\frac{1}{R^2}$.

The difficult part of the previous algorithm was in automating the decision as to whether the SVD curves were scaling linearly versus higher order. By using the whitening transformation, we have constructed a standardization of the slopes, and have obtained a model set to which comparisons can be made. Furthermore, in determining which dimensions are scaling properly, we need make only 1 computation, at a distance of $R/2$ from center, where the distinction between linear and higher order scaling is the greatest. Let σ_i denote the n singular values at a distance of $R/2$ from center. Then the decision is thus:

If $\sigma_i > \frac{3}{8}$ dimension i is in the span of the tangent plane. Otherwise, dimension i is not in the span.

Algorithm 3.4 *The Modified LSVD with Fast Decision Algorithm*

- Compute V, S from the KL of X.
- Sort the data in X in ascending order.
- Compute $Y = XV(\sqrt{p}S^{-1})$. This is also the noise reduction step. In computing S^{-1}, one may choose to keep only those normalized singular values above a noise floor.

- Let r be the radial distance from the step above that is closest to center. Let R be the maximum radial distance. The cutoff value, C, for the dimensionality will be if the singular value is greater than the midpoint between the linearly scaled singular values and the quadratically scaled singular values:

$$C = \frac{1}{2}\left(\frac{r}{R} + \frac{r^2}{R^2}\right)$$

- Compute the singular values for the data within r of center. If this is the first j values, then we compute the singular values as $\sqrt{S}$, where

$$\Phi S \Phi^{\mathrm{T}} = \left(\frac{1}{j}\right) Y(1:j,:)^{\mathrm{T}} Y(1:j,:)$$

- Determine which of the singular values, $\sqrt{s_{ii}}$, are greater than C. This is the local estimation of the dimension of the tangent space.

3.6 Clustering Examples using the SVD and Modified SVD

Standard v. Modified SVD on Non-Uniform Data

This example shows the difference between the estimated linear scalings of a non-uniformly distributed data set using the standard approach, versus the new approach which whitens the data.

The data is again the north pole cap of the unit sphere, with data non-uniformly distributed. The image to the left in Figure 3.2 is the estimation using the standard approach. The image to the right depicts the singular values and the analytic estimates of the whitened data.

Here we see that the original SVD curves have differing slopes than the analytic estimate, since the distribution has changed from being uniform. On the other hand, we are able to predict the scalings for the locally whitened data. This is significant if we are to automate the algorithm.

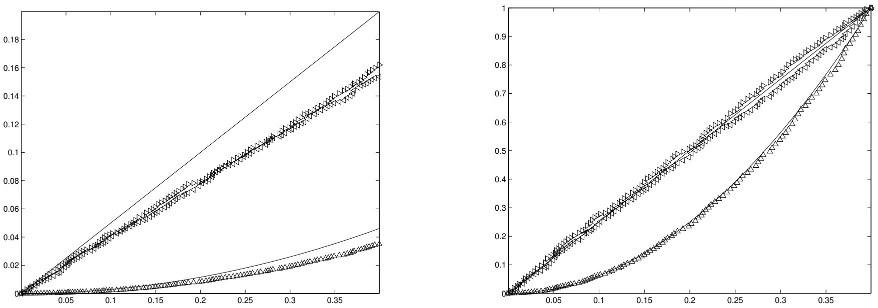

Figure 3.2: SVD Curves for the Unit Sphere. Numerical values are dotted, analytic estimates are solid. On the left, is the standard SVD computations, and on the right are the whitened computations, with the analytic estimates.

The Lorenz Attractor

In this example, we consider a set of points obtained by aging a numerical solution to the Lorenz Equations:

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= rx - y - xz \\ \dot{z} &= xy - bz \end{aligned} \tag{3.8}$$

with $\sigma = 10$, $b = \frac{8}{3}$, and $r = 28$.

The attracting set is shown in Figure 3.3. We randomly selected points of the attracting set and computed the local dimensionality. Specifically, we chose 500 points of the attractor at which to do the computation, and each local data set retained 200 points. The full data set consisted of 9000 points. Other methods of computing dimensionality have placed the correlation dimension at approximately 2.05 (for example, see [30]). We obtained approximately 73% of the clusters returning two dimensions, and 27% returning three dimensions.

If we interpret these results to mean that approximately 73% of the manifold is two dimensional and approximately 27% of the manifold is three dimensional, then overall,

$$(2)(.73) + (3)(0.27) \approx 2.27$$

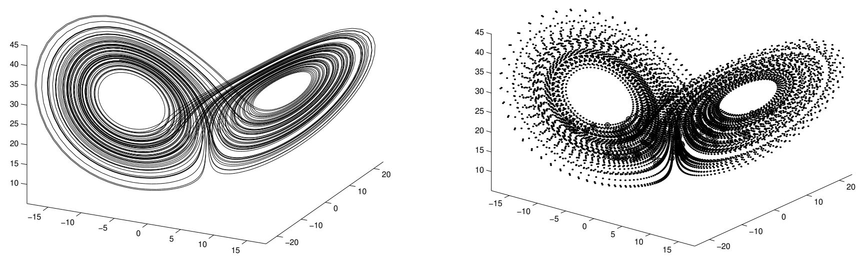

Figure 3.3: The original data representing the Lorenz Attracting Set

We expect our value to be somewhat higher since we are estimating the dimension of an approximate manifold for the data, rather than the data itself.

3.7 Conclusions

KL has played a dominant role in the search for coherent structures in turbulence. First proposed in [53] and subsequently in [54, 2, 9, 8, 7, 66, 68] and others, the question of how (and if) coherent structures and their local interactions play a role in turbulence is still a major source of research.

The KL expansion has a long history of being used for dimensionality estimation and construction of optimal coordinate systems. We have presented current methods which rely on the KL expansion as the basis for the local and global dimensionality algorithms, culminating in the Modified Local SVD algorithm for the local automated deduction of dimensionality. We have also presented a novel application of using this local dimensionality estimate as one aspect to characterizing the complexity of data.

The main feature of the algorithm is in the ability to locally whiten the data before estimating dimension, thereby removing the dependence of the scaling of the singular values to the distribution of the data. Intrinsic to the local whitening is also a local filter that removes small amounts of noise from the data.

This algorithm has shown to be an optimal choice when using the neural charts routine on data that represents a well-defined local manifold. We will see in the examples, that in the case that a local manifold is not well-defined by the data, the algorithm estimates the encapsulating dimension.

Now that the neural charts algorithm has its projection maps, we are ready to turn to inverting those maps. The main tool of choice is the neural network.

Chapter 4

FUNCTION APPROXIMATION USING NEURAL NETWORKS

4.1 Introduction

Key to the success of the neural charts algorithm is the construction of the nonlinear inverse to the local projections constructed in Chapter 3. We will see in the examples that essential to maintaining the original dynamics after compression is the smooth reconstruction of the manifold on which the dynamics lie. In fact, in one of our examples, a dynamical system that was dense on the manifold originally becomes locked into a periodic orbit after only a linear reconstruction.

As was stated in Chapter 1, we have chosen neural function approximation over other methods (i.e., polynomial approximation) because they are relatively easy to implement, they provide robust approximations with respect to noise (so that they generalize well), and they are more adept at overcoming the curse of dimensionality [4, 5].

The goal of this chapter is to first give a short introduction to the subject of Neural Networks (and more specifically, Neural function approximation), then give some adaptations we have made and show how the neural approximators are incorporated into the neural charts algorithm.

4.2 The Multilayer Perceptron

The Multilayer Perceptron (or three layer feedforward neural network) has been the workhorse of the function approximation toolbox. Before getting into the details of the function approximation model and training, some remarks are in order.

At the turn of the century, D. Hilbert compiled a list of 23 unsolved problems as a challenge for twentieth century mathematicians. Hilbert's thirteenth problem concerns the issue of whether functions of several variables can be represented in terms of superpositions of continuous functions of fewer variables. This problem was solved by Kolmogorov in 1957 when he showed that, over a compact domain, functions of several variables could be approximated by relatively small superpositions of functions of one variable. Although the neural network is not the function of one variable he supplied, the neural approximation does provide a solution to Hilbert's thirteenth problem. It is interesting to note that here, at the end of the twentieth century, this solution to a problem posed at the end of the last century is the driving force behind many discoveries extending into the twenty first century. For more on the connections from neural networks to Kolmogorov's solution, see [10].

We will first motivate the form of the neural approximation by building a simplistic mathematical model of the biological neuron. We will quickly depart from the biology, simply considering the model as a computational unit. It is impressive that from such a simple model, we will be able to construct a "Universal Function Approximator"!

4.2.1 From the Biological to the Symbolic

The neuron is the basic building block of the central nervous system. There are many different types of neurons, but each can be anatomically broken down into three parts: the dendrites, the cell body, and the axon. Information flows from the dendrites to the cell body through the axon to a synaptic junction connecting to the dendrite to the next neuron.

The synaptic junction is made up of the presynaptic node (the end of an axon), the postsynaptic node (the beginning of a dendrite), and the "empty space", which is the synapse.

For the purposes of our basic mathematical model, we will assume the following processing features that a neuron makes to information (we will refer to Neurons as Nodes in what follows):

1. As information flows across a synapse, information can be amplified, inhibited, or re-polarized. If we let x denote the signal, then synaptic processing is represented as a multiple of x:

$$x \to wx$$

   where w will be referred to as a *weight*. Since there are multiple incoming signals, we denote by w_{ij} the weight connection from Node j to Node i.

2. The information flow across the dendrite to the cell body is lossless.

3. At the cell body, information from across the dendrites is collected and processed in the following way.

   (a) The incoming signals are summed. Let $j = 1..m$ denote the index of the signal coming to Node i:

$$\sum_{j=1}^{m} w_{ij}x$$

   (b) This quantity is added to the resting state of the cell, which is assumed to be constant:

$$\sum_{j=1}^{m} w_{ij}x + b_i$$

   and b_i is called the bias of the Node. The summed quantity is referred to as the Prestate of the Node.

(c) Biologically, we assume an "all or nothing" response to the incoming signal. That is, if the signal strength is above a certain threshold, the neuron fires. If not, then the information is not passed along. Mathematically, we use a step function, with the step at 0. This is not a differentiable function, however. We substitute for the step function a "sigmoidal" function, $\sigma(x)$. This is defined to be a monotonically increasing function so that

$$\lim_{x \to -\infty} \sigma(x) = A \qquad \lim_{x \to \infty} \sigma(x) = B$$

where A and B are constants. Common choices for the sigmoidal function include:

i. The inverse tangent:

$$\sigma(x) = \mathrm{atan}(x)$$

ii. The exponential sigmoidal function:

$$\sigma_\beta(x) = \frac{1}{1 + \mathrm{e}^{-\beta x}}$$

We will use the latter choice, since the computation of its derivative is especially easy to implement on the computer:

$$\sigma'_\beta(x) = \frac{\beta \mathrm{e}^{-\beta x}}{1 + \mathrm{e}^{-\beta x}} = \beta \sigma(x)(1 - \sigma(x))$$

Other functions that transfer the incoming signal out to the axon are possible. At times, we will use the following, denoted by $t(x)$ as transfer functions:

i. The Linear Node

$$t(x) = x$$

ii. The Circular Node (two inputs, two outputs per node):

$$t(x, y) = \left(\frac{x}{\sqrt{x^2 + y^2}}, \frac{y}{\sqrt{x^2 + y^2}} \right)$$

iii. The Spherical Node (three inputs, three outputs):

$$t(x, y, z) = \left(\frac{x}{\sqrt{x^2 + y^2 + z^2}}, \frac{y}{\sqrt{x^2 + y^2 + z^2}}, \frac{z}{\sqrt{x^2 + y^2 + z^2}} \right)$$

We will discuss the last transfer function in detail when we apply the neural networks to a triangulation problem.

4. The axon sends along the information to the synapse.

In summary, we have the following information processing for the i^{th} single node:

$$x \to \underbrace{\sum_{j=1}^{m} w_{ij} x + b_i}_{\text{Prestate}} \to \underbrace{\sigma \left(\sum_{j=1}^{m} w_{ij} x + b_i \right)}_{\text{State}}$$

In vector form,

$$x \to (\boldsymbol{w}_i)^{\mathrm{T}} x + b_i \to \sigma \left((\boldsymbol{w}_i)^{\mathrm{T}} x + b_i \right)$$

So that, taking $\boldsymbol{x} \in \mathbb{R}^N$, and having k Nodes,

$$\boldsymbol{x} \to \left((\boldsymbol{W}^{(0)})^{\mathrm{T}} \boldsymbol{x} + \boldsymbol{b} \right) \to \sigma \left((\boldsymbol{W}^{(0)})^{\mathrm{T}} \boldsymbol{x} + \boldsymbol{b} \right)$$

where

$$\boldsymbol{W}^{(0)} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k) \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{pmatrix}$$

and σ operates on a vector or matrix component-wise. Also note that, if σ were a linear function, then the neural network would be a linear operator.

4.2.2 The Three Layer Feedforward Neural Network

The usual method of describing a neural network's architecture is to use a connected graph with computational nodes. The basic feedforward neural network uses three layers of "nodes". The first layer is the input layer, so the number of nodes is equal to the dimension of $\boldsymbol{x}$, and the transfer function is linear. The hidden layer contains the nonlinear transfer function, and there is a choice as to how many nodes one includes in the hidden layer. We assume there are k nodes in the hidden layer. The output layer again has a linear transfer function, and there are as many nodes here as there are output variables. Each edge of the graph is weighted, and we refer to edges from the input layer to the hidden layer as the matrix

$$\boldsymbol{W}^{(0)} = \left[W_{ij}^{(0)}\right]_{i=1:n(1),\, j=1:n(0)}$$

where $n(i)$ is the number of nodes in Layer i, and

$$\boldsymbol{W}^{(1)} = \left[W_{ij}^{(0)}\right]_{i=1:n(2),\, j=1:n(1)}$$

so that W_{ij}^{k} refers to the weight in the k^{th} layer, going from Node j in the k^{th} layer to Node i in the $k+1^{\text{st}}$ layer.

There are also vectors of biases associated with the hidden layer, and output layer, respectively:

$$\boldsymbol{b}^{(0)} \text{ and } \boldsymbol{b}^{(1)}$$

For the purposes of the backpropagation algorithm given later, note that the biases can also be represented as one additional node in the input and hidden layers, with constant transfer function, $\sigma(x) = 1$. We will adopt this notation, and suppress the bias specification in Section 4.2.1. Figure 4.1 shows the graphic layout of the three layer, feedforward neural network.

The purpose of this network is to provide an approximation to an arbitrary continuous function, $\boldsymbol{F}$, defined on a compact domain X, and we will use the

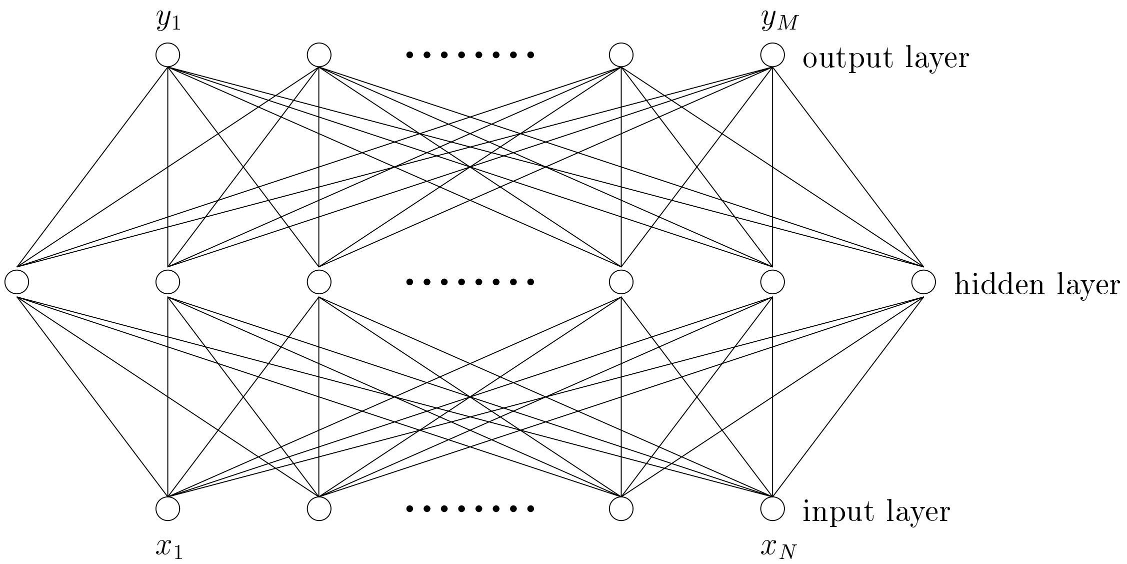

Figure 4.1: The neural network architecture.

notation $\mathbf{y} = \boldsymbol{F}(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^M$. The network will render the approximation in terms of a sigmoidal basis so that the approximation to $\boldsymbol{F}$ is given pointwise by:

$$\boldsymbol{y}^{(i)} = \boldsymbol{F}(\boldsymbol{x}^{(i)}) \approx \left(\mathbf{W}^{(1)}\sigma\left(\mathbf{W}^{(0)} \cdot \mathbf{x}^{(\mathbf{i})} + \mathbf{b}^{(0)}\right) + \mathbf{b}^{(1)}\right) \tag{4.1}$$

In [16] it is shown that sums of the form in equation (4.1) are dense in the space of continuous functions defined on the unit cube. In fact, there are more general results that can be found in [35, 36, 33, 34, 26, 27] and more specifically, they show that a neural approximation not only approximates the function, but can also approximate the first m derivatives using the same functional parameters.

Training and Error

We have now seen how the structure of a neural network can be exploited as to interpret the network as a function approximator. For a fixed value of the parameters in the matrices $\boldsymbol{W}^{(i)}$ and biases $\boldsymbol{b}^{(i)}$, the network outputs a smooth function in $\boldsymbol{x}$. As mentioned previously, we adopt the notation of including the biases in the parameters $\boldsymbol{W}^{(i)}$, and add an extra node on the hidden and input layers.

Define the mean square error of the approximation, given a set of parameters $\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}\}$, as:

$$E(\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}) = \frac{1}{2|X|} \sum_{i=1}^{|X|} \|F(\mathbf{x}^{(\mathbf{i})}) - \left(\mathbf{W}^{(1)}\, \sigma \left(\mathbf{W}^{(0)} \cdot \mathbf{x}^{(\mathbf{i})}\right)\right) \|_2^2 \tag{4.2}$$

where $|X|$ denotes the number of points in the data set X. Then E is a smooth function of the weights and biases, so that (in theory) we can find its minimum. From the method of steepest descent, we update the weights:

$$W_{jk}^{(i)}(t+1) = W_{jk}^{(i)}(t) + \alpha(t)d(t) \tag{4.3}$$

until the error has become smaller than a prescribed value. For the standard gradient descent, for example, we have

$$d(t) = \frac{\partial E}{\partial W_{jk}^{(i)}}(t)$$

The process of updating weights in an iterative fashion to obtain the optimal parameter values is referred to in the neural network literature as network training. How we find the value of $\alpha(t)$ and the definition of $d(t)$ is where the divergence in algorithms comes into play. There are many algorithms built especially for this purpose, such as Line Minimization. Other update methods can be applied, since training is a general, unconstrained optimization problem:

$$\min_{\boldsymbol{W}^{(0),(1)}} \quad E$$

Most optimization algorithms will require that we obtain expressions to compute the partial derivatives of E with respect to the weights and biases. It is fortunate that, for large networks, there is a recursive algorithm to accomplish this. This will be discussed in the next section. First, we introduce some notation that will make the algorithm more clear.

Define the prestate for layer i, Node j at time t:

$$P_j^i = \sum_{k=1}^{n(i-1)} W_{jk}^{i-1} S_k^{i-1}, \text{ for } \ i = 1 \dots L$$

where S_j^i is the State of Node j on layer i, which is

$$S_j^i = \begin{cases} \sigma\left(P_j^i\right) & i > 0 \\ x & i = 0 \end{cases}$$

Backpropagation of Error

We now detail a recursive algorithm, known as the backpropagation of error [3], which gives an efficient method for computing the partial derivatives for training (see Equation (4.3)).

The goal is to compute

$$\frac{\partial E}{\partial W_{jk}^{i-1}}$$

for each i, j, k.

From the chain rule, we can write

$$\frac{\partial E}{\partial W_{jk}^{i-1}} = \frac{\partial E}{\partial P_j^i} \cdot \frac{\partial P_j^i}{\partial W_{jk}^{i-1}}$$

which we can re-express, using:

$$\frac{\partial P_j^i}{\partial W_{jk}^{i-1}} = S_k^{i-1} \quad \frac{\partial P_j^{i+1}}{\partial P_k^i} = W_{jk}^i \sigma'(P_k^i)$$

and defining:

$$\delta_j^i = \frac{\partial E}{\partial P_j^i}$$

Then we have:

$$\frac{\partial E}{\partial W_{jk}^{i-1}} = \delta_j^i \cdot S_k^{i-1}$$

So, we need an algorithm to compute δ_j^i.

For the output layer (L), with aribtrary transfer function σ,

$$\delta_j^L = \frac{\partial E}{\partial P_j^L} = \frac{\partial E}{\partial S_j^L}\frac{\partial S_j^L}{\partial P_j^L} = \sigma'(P_j^L)\frac{\partial E}{\partial S_j^L}$$

where $\frac{\partial E}{\partial S_j^L}$ is computed from the error by substituting S_j^L into Equation (4.2):

$$E = \frac{1}{|X|}\sum_{i=1}^{2|X|} \|\boldsymbol{y}^{(i)} - \begin{pmatrix} S_1^{(L)} \\ \vdots \\ S_m^{(L)} \end{pmatrix} \|_2^2$$

which implies that:

$$\frac{\partial E}{\partial S_j^L} = \frac{-1}{|X|}\sum_{i=1}^{|X|} |y_j^{(i)} - S_j^L| \tag{4.4}$$

For the hidden nodes,

$$\delta_j^i = \frac{\partial E}{\partial P_j^i} = \sum_{k=1}^{n(i+1)} \frac{\partial E}{\partial P_k^{i+1}}\frac{\partial P_k^{i+1}}{\partial P_j^i}$$

Noting $\frac{\partial P_k^{i+1}}{\partial P_j^i} = W_{kj}^i \sigma'(P_j^i)$, we obtain:

$$\delta_j^i = \sigma'(P_j^i)\sum_{k=1}^{n(i+1)} \frac{\partial E}{\partial P_k^{i+1}} W_{kj}^i$$

Which, from our definition of δ_j^i, is recursively defined as:

$$\delta_j^i = \sigma'(P_j^i)\sum_{k=0}^{n(i+1)} \delta_k^{i+1} W_{kj}^i \tag{4.5}$$

This can be interpreted as a backpropagating the δ_j^i backwards through the network. The algorithm is summarized as:

Algorithm 4.1 *The Backpropagation Algorithm*

1. Forward propagate all the x's, keeping track of the error.

2. Compute δ_j^L, for $j = 0 \ldots M-1$ using:

$$\delta_j^L = \sigma'(P_j^L)\frac{\partial E}{\partial S_j^L}$$

3. Compute all other δ's:

$$\delta_j^i = \sigma'(P_j^i) \sum_{k=0}^{n(i+1)} \delta_k^{i+1} W_{kj}^i$$

4. Compute:

$$\frac{\partial E}{\partial W_{jk}^{i-1}} = \delta_j^i \cdot S_k^{i-1}$$

for $i = 1 \ldots L$, $j = 1 \ldots n(i+1)$, and $k = 1 \ldots n(i)$.

4.2.3 Autoassociation Neural Networks

In the previous section, we saw that a three layer feedforward neural network is capable of approximating any continuous function on a compact domain.

Suppose now that we want to reduce the dimensionality of the data set X. This can be expressed as constructing homeomorphisms

$$G : X \subset \boldsymbol{R}^N \rightarrow Y \subset \boldsymbol{R}^M$$
$$H : Y \subset \boldsymbol{R}^M \rightarrow X \subset \boldsymbol{R}^N$$

We could build two separate neural networks, one for each function, if we knew a priori the target values in the reduced dimensional space Y. For the moment, suppose we are not concerned with the form of the reduced representation. Then the compression paradigm can be reformulated: find functions G and H as before, but we only require that the composition $G \circ H$ is the identity on X. We envision this composition as a single, large network composed of two, three layer networks glued together, as shown in Figure 4.2.

In this figure, the input and output layers have N nodes, the bottleneck layer has $M << N$ nodes, and the two hidden layers have some pre-determined number of nodes. It has been shown in [16, 27, 63] that these hidden layers are necessary to perform a nonlinear compression and decompression. That is, without the nonlinear hidden layer, this network performs a linear dimensionality reduction- for which we know the optimal solution is given by its Karhunen-Loève expansion.

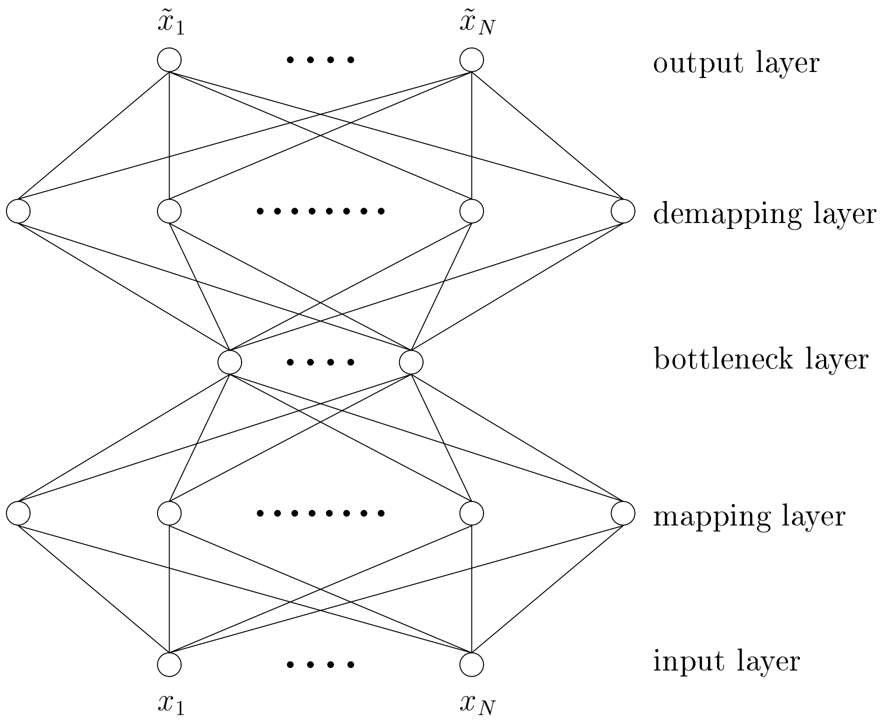

Figure 4.2: The bottleneck network architecture.

This network has been widely used as a compression technique, and in a recent work [48] several criteria were included in the training algorithm to force certain specific representations on the bottleneck layer.

In the last chapter, this network will be discussed as a global nonlinear dimensionality reducing model (also see Table 1.1).

4.2.4 Spherical Nodes and Triangulation

One of the advantages of approximating functions by feedforward neural networks is that the approximation may be viewed as *model free*. However, the enormous flexibility provided by this approach may also be a serious handicap given that the training is developed purely on a finite collection of input/output pairs which may not all be known simultaneously, or may be insufficient in number. It is clear that any a priori modeling of the problem which is incorporated into the network will serve to mitigate data shortages and improve approximations.

Developing models is a challenging problem and this is, afterall, the purpose of the neural network. Nonetheless, it is often possible to assist in the development of the model by incorporating, or *hard-wiring* certain types of mapping capability. For instance, in standard implementations of feedforward networks, valuable information concerning the topology of the data set is often simply ignored. Consider that the standard neural processing unit, the sigmoidal function, encodes information on the open interval, e.g., $(-1, 1)$. If the raw data is homeomorphic to a different manifold, such as a circle or a sphere, then the network does not model this directly, but rather in terms of a Cartesian product of open intervals.

Hence, our construction of a spherical node is motivated by the desire to incorporate a spherical manifold model into the network which is appropriate for modeling raw data that is topologically equivalent, i.e., homeomorphic to a sphere. Previous work [50] has shown that a a topologically circular node is useful for encoding periodic information. This paper extends these ideas to data with spherical topology.

The implementation of the spherical node can be done efficiently within the framework of the standard backpropagation algorithm, and the changes needed are summarized below.

The main application of the spherical node in this section will be to the triangulation of a data clouds representing sampled contours obtained from 3D images of the lobes of the lungs. We use the spherical node in a bottleneck architecture to approximate a mapping from the data cloud to the sphere as well as the mapping from the sphere to the lung. The smooth approximation is then refined using a modified topology preserving mapping [51] to produce a sharp triangulation of the data on the lung.

Implementation of the Spherical Node

The spherical node can be envisioned as a processing unit with three input sites and three output sites; each input site collects data from the previous layer, and each output site transmits the location of the spherical projection to the next layer. For computational purposes, the spherical node is implemented as a triplet of standard processing nodes, each with a single input and output site. Before developing the algorithm we will define our notation.

The neural network has L layers. In the most general case, each layer consists of nodes, which may be spherical, circular, sigmoidal or linear. We include the computations for the standard sigmoidal transfer function $\sigma(x)$ for comparison purposes. See [50] for further information on implementation of the circular node.

The j^{th} node in layer i will be denoted by $\mathcal{N}_j^{(i)}$. The number of nodes in layer i is $n(i)$, and for fixed i the nodes $\mathcal{N}_j^{(i)}$ are indexed from $j = 1$ to $j = n(i)$. The state value of node $\mathcal{N}_j^{(i)}$ will be denoted by $S_j^{(i)}$ and is obtained by applying a transfer function to the prestate value $P_j^{(i)}$. A spherical node will occur as a triplet of linked nodes denoted $\{\mathcal{N}_j^{(i)}, \mathcal{N}_{\gamma(j)}^{(i)}, \mathcal{N}_{\gamma^2(j)}^{(i)}\}$. The defining quality of a spherical node is given by the constraint equation

$$(S_j^{(i)})^2 + (S_{\gamma(j)}^{(i)})^2 + (S_{\gamma^2(j)}^{(i)})^2 = 1.$$

It is requiring that this equation be satisfied which forces the data at the nodes $\{\mathcal{N}_j^{(i)}, \mathcal{N}_{\gamma(j)}^{(i)}, \mathcal{N}_{\gamma^2(j)}^{(i)}\}$ to lie exactly on a sphere.

This constraint equation fits naturally into the backpropagation algorithm as discussed below.

Forward Propagation

The prestate values, P_j^i at each node $\mathcal{N}_j^{(i)}$ are computed in the same manner for all node types, i.e.,:

$$P_j^i = b_j^{(i)} + \sum_{k=1}^{n(i-1)} W_{kj}^{(i-1)} S_k^{(i-1)}.$$

The final state values are then determined by the prestate values using different formulas depending on the type of node:

$$S_j^i = \begin{cases} P_j^i / \sqrt{(P_j^i)^2 + (P_{\gamma(j)}^i)^2 + (P_{\gamma^2(j)}^i)^2} & \text{if } \mathcal{N}_j^{(i)} \text{ is spherical} \\ \\ \sigma_j^{(i)}(P_j^i) & \text{if } \mathcal{N}_j^{(i)} \text{ is sigmoidal} \end{cases}$$

It is convenient to define the *radial value*. Let

$$R_j^{(i)} = \sqrt{(P_j^i)^2 + (P_{\gamma(j)}^i)^2 + (P_{\gamma^2(j)}^i)^2}$$

Note that in the spherical node, $R_j^{(i)} = R_{\gamma(j)}^{(i)} = R_{\gamma^2(j)}^{(i)}$.

Error and Gradient Computations

Given an output state value S_j^L, the associated desired goal value is denoted y_j, where $j = 1, \ldots, n(L)$. The total squared error E in the network for this state S is then defined as

$$E = \frac{1}{2} \sum_{j=1}^{n(L)} (S_j^L - y_j)^2.$$

As usual, if more than one state is used to form an average error, the gradient of the error simply sums over all these states; hence to avoid using an index to indicate the state, we develop the gradient computations for a single state only; the reader may then take the appropriate sum for an average error over many states. To develop the formulas for the back-propagation algorithm, we compute the gradient of the error with respect to the weights and biases. Hence we compute the values

$$\frac{\partial E}{\partial b_j^{(i)}} \text{ and } \frac{\partial E}{\partial W_{jk}^{(i)}}$$

for all i,j, and k. For notational convenience, let us define the following:

$$\boldsymbol{\nabla}_{\mathcal{S}}(E_j^{(i)}) \equiv \left(\frac{\partial E}{\partial S_j^{(i)}}, \frac{\partial E}{\partial S_{\gamma(j)}^{(i)}}, \frac{\partial E}{\partial S_{\gamma^2(j)}^{(i)}} \right)$$

and

$$\boldsymbol{\nabla}_{P_j^{(i)}}(\mathcal{S}) \equiv \left(\frac{(P_{\gamma(j)}^{(i)})^2 + (P_{\gamma^2(j)}^{(i)})^2}{(R_j^{(i)})^3}, \frac{-P_j^{(i)} P_{\gamma(j)}^{(i)}}{(R_j^{(i)})^3}, \frac{-P_j^{(i)} P_{\gamma^2(j)}^{(i)}}{(R_j^{(i)})^3} \right)$$

Standard application of the chain rule for derivatives leads back to the formulas:

$$\frac{\partial E}{\partial b_j^{(i)}} = \begin{cases} \boldsymbol{\nabla}_{\mathcal{S}}(E_j^{(i)}) \cdot \boldsymbol{\nabla}_{P_j^{(i)}}(\mathcal{S}) & \text{if } \mathcal{N}_j^{(i)} \text{ is spherical} \\ \\ \frac{\partial E}{\partial S_j^{(i)}}(\sigma_j^{(i)})'(P_j^{(i)}) & \text{if } \mathcal{N}_j^{(i)} \text{ is sigmoidal} \end{cases}$$

and

$$\frac{\partial E}{\partial W_{jk}^{(i)}} = \begin{cases} \left(\boldsymbol{\nabla}_{\mathcal{S}}(E_k^{(i+1)}) \cdot \boldsymbol{\nabla}_{P_k^{(i+1)}}(\mathcal{S})\right) S_j^{(i)} & \text{if } \mathcal{N}_k^{(i+1)} \text{ is spherical} \\ \\ \frac{\partial E}{\partial S_k^{(i+1)}}(\sigma_k^{(i+1)})' \left(P_k^{(i+1)}\right) S_j^{(i)} & \text{if } \mathcal{N}_k^{(i+1)} \text{ is sigmoidal.} \end{cases}$$

What remains then is to compute the derivatives $\partial E/\partial S_j^{(i)}$, which is done recursively. At the start of the the recursion, we have

$$\frac{\partial E}{\partial S_j^L} = S_j^L - y_j.$$

For other "lower" layer values, we have

$$\frac{\partial E}{\partial S_k^{(i-1)}} = \sum_{j=0}^{N^{(i)}-1} \frac{\partial E}{\partial S_j^{(i)}} \frac{\partial S_j^{(i)}}{\partial S_k^{(i-1)}}. \tag{4.6}$$

where

$$\frac{\partial S_j^{(i)}}{\partial S_k^{(i-1)}} = \begin{cases} \boldsymbol{\nabla}_{P_j^{(i)}}(\mathcal{S}) \cdot \left(W_{kj}^{(i-1)}, W_{k\gamma(j)}^{(i-1)}, W_{k\gamma^2(j)}^{(i-1)}\right) & \text{if } \mathcal{N}_j^{(i)} \text{ is spherical} \\ \\ (\sigma_j^{(i)})'(P_j^{(i)}) W_{kj}^{(i-1)} & \text{if } \mathcal{N}_j^{(i)} \text{ is sigmoidal.} \end{cases}$$

With these modifications, the back-propagation algorithm proceeds as usual.

Application to a Surface of Revolution

The following is an example of how the autoencoder with a spherical node may be used to compress data in R^3. The network used a 3-10-1-10-3 architecture with all sigmoidal nodes with the exception of the spherical node at the bottleneck. We presented the network with data sampled from a surface of revolution (a cubic polynomial in x, rotated about the x-axis). The left picture of Figure 4.3 shows

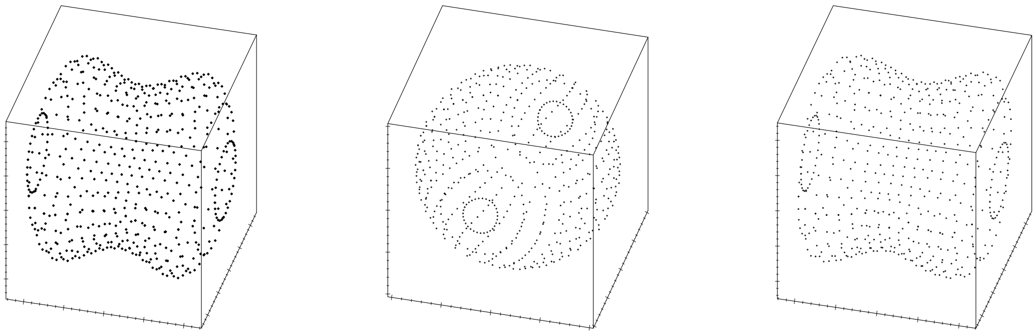

Figure 4.3: The spherical node representation of the surface of revolution. Left: Original Data. Center: Data on the Bottleneck. Right: Reconstructed Data.

the original data set (and the goal data set). Training resulted in a MSE of approximately 0.0008. The center picture in Figure 4.3 shows the data on the spherical bottleneck layer. The last picture shows the reconstruction in R^3. We note that the autoencoder will (i) choose the spherical representation, and (ii) gives two independent mappings, one from R^3 to S^2, and one from S^2 to R^3. We will exploit these features in solving the triangulation problem.

Application to the Triangulation of Data Sets

In this section, we show how the auto-encoder, together with a modified topology preserving map, can triangulate a data set that is homeomorphic to the sphere S^2.

Triangulation of a data set can be thought of as imposing an ordering on the data so that it can be grouped as the points of a polygon. This ordering is essential for graphics drivers to render the data as a surface. Once triangulated, the graphics driver can then apply texturing and lighting to the surface. Because of the texture application, it is important to get as uniform a triangulation as possible.

There are many approaches to solving the triangulation problem, most of which are combinatorial on the exact data given. We will take a different approach.

The data we use was generously loaned to us from the Vesalius project at C.S.U. (currently Glaxo Virtual Anatomy), which is involved with getting a complete, three dimensional computer rendering of human anatomical structures. The data was obtained by thinly slicing a human cadaver, then sampling and sorting the data from each slice. The lung data that we use below can thus be thought of as a contour plot of the surfaces of the five lobes of the lung (three lobes on the right, two on the left). In order to convert this raw data into surfaces (upon which one can apply textures, etc.), the data must first be triangulated.

We will triangulate the data in the following way. Suppose we have a generalized mapping from S^2 to our data. We can then triangulate the sphere (which is easy to do), then transport the triangulation through the mapping to the data set. We also show how the topology preserving map (TPM) can be used to get an even better fit to the data. Note that this paradigm must involve a method to decrease the error on individual points, as the resulting surfaces must fit together to form a complete lung.

We begin by allowing the autoencoder to choose the spherical representation of the data. Figure 4.4 shows the result of training a 3-25-1-25-3 network (again, all sigmoidal nodes except for the spherical node at the bottleneck). The picture to the left shows the original data from the right middle lobe of the lung. The center picture shows the autoencoder's choice of the spherical representation, and the last picture shows the reconstruction (MSE: 0.017). We now separate the autoencoder so that the mapping we want is formed by the 1-25-3 portion of the network, this is the mapping from S^2 to R^3.

Some words of caution are appropriate here: (i) because of the smoothness of the transfer functions, the reconstruction will necessarily be smooth (unless we increase the number of hidden nodes to an unreasonable level), and (ii) allowing

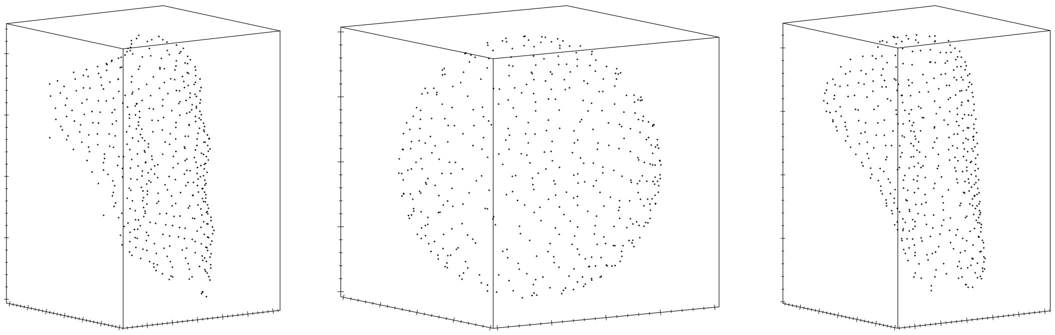

Figure 4.4: Training on the right middle lobe of the lung. To the left is the original data set, the center represents the data on the spherical bottleneck, and the last image is the neural reconstruction.

the autoencoder to choose the spherical representation may result in a mapping of only a partial sphere to the data set.

Figure 4.5 shows such a choice, where a 3-25-1-25-3 network was trained using the data from the left lower lobe of the lung. This lobe presents many difficulties for the autoencoder because of the "sharp" edges and large fold. The picture to the left is of the original data set. The center picture shows that the autoencoder's choice of spherical representation is only half the sphere. Note in the reconstruction (the picture to the left), the bottom of the surface is missing, although numerically we were able to get a MSE of 0.048.

The autoencoder has given us a 1-1 correspondence between the spherical points and the lung data. We will maintain this correspondence, but we will change the spherical representation off-line so that the points are uniformly distributed. This was accomplished by an algorithm that used a normalized version of Coloumb's Law. Once the domain of the network has been uniformly distributed, we train the 1-25-3 network. Figure 4.6 shows the results. The mapping is now from all of S^2 (which is essential for transporting a triangulation) to an approximation of the lobe of the lung. The picture to the left in Figure 4.6 shows the

Figure 4.5: Left: The original data from the left lower lobe of the lung. Center: The network has chosen a non-uniform distribution of points on the sphere, due to the presence of the diaphragm. Right: The reconstruction.

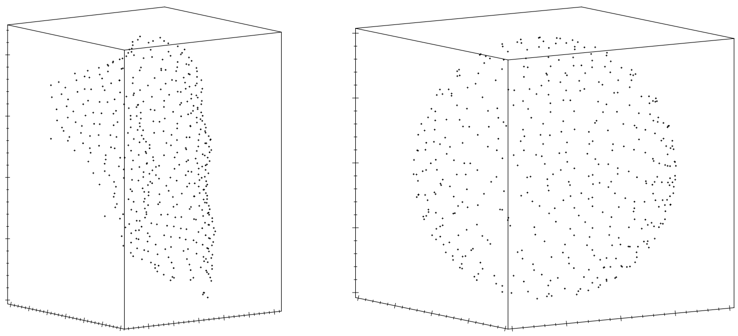

Figure 4.6: Forcing a uniform distribution over the sphere.

result of uniformization, and the picture to the right is the new reconstruction. The global characteristics of the data are now better represented (i.e., the bottom of the surface is there), but the error has increased significantly (MSE is now approximately 0.17).

We need an algorithm to take us the extra step; that is able to squeeze our smooth generalization into the corners of the data set. In other words, a mapping that (i) preserves the topological ordering that we will place on a triangulation of the sphere, and (ii) pulls the approximation into the data set.

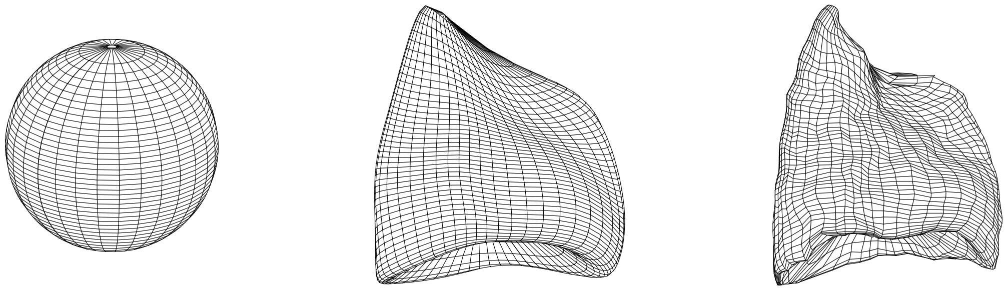

Figure 4.7: Final Mapping of the Sphere to the Lung using Kohonen's SOM.

Generally, one uses the SOM to map an indexed set of vectors into some fixed data set. The index set is usually a two dimensional lattice, but this is just a convenience and not a necessary condition for the algorithm. The power of the SOM is that we can take any set of n-tuples to be the index set, and we can arrange the n-tuples any way we like. The algorithm will then attempt to transport the ordering we have imposed into the fixed data set. We run into technical difficulty if we completely generalize the algorithm (for example, if we begin the algorithm by randomly initializing the set of vectors), but if we use as our index set the generalized lung data, indexed by the triangulation, then we will require the algorithm to simply *maintain* the ordering, and pull the data points only a short distance. In essence, we are using the neural network to give the topology preserving map a good set of initial values, thus ensuring convergence. In Figure 4.7, we show the initial triangulation of points on the sphere, and the center picture shows the result of transporting this triangulation to an approximation of the lung data. It is this set of points which we use as the initial weights for the SOM. The last picture in Figure 4.7 shows the result of applying the SOM to the initial weights. Note that we have been able to significantly reduce the error (arithmetic mean, -0.000215; MSE, 0.001).

Summary

We have defined the spherical node as a means to model data which is topologically spherical. It consists of 3 processing units whose values are constrained to lie on the sphere. We show how this definition may be implemented in the standard back-propagation algorithm and present the required formulae for the derivates. The spherical node is then used at the bottleneck of the the nonlinear auto-encoder architecture to solve a generalized triangulation problem. Our hybrid approach combines a smooth topological model for the data which is refined in a post-processing phase using a modified topology preserving mapping. The error of the triangulation was reduced to an essentially negligible level.

4.3 Radial Basis Functions

The interest in radial basis functions has surged since the mid to late 1980's when some remarkable properties of its interpolation matrix had come to light, and the connection between interpolation and adaptive learning machines had been shown. This section begins with a review of the seminal papers by Powell [64] (who also includes the important work by Micchelli [59]), and from Broomhead and Lowe [13]. Following the historical sequence, we first examine the interpolation problem, followed by the current use of radial basis functions as a type of neural network.

We start with a definition of the *euclidean distance matrix*:

Definition 4.1 *The Euclidean Distance Matrix*

Let $\{\boldsymbol{x}^{(i)}\}_{i=1}^{P}$ be a set of points in $\mathbb{R}^n$. Let the matrix $\boldsymbol{A}$ be the P by P matrix so that the $(i,j)^{\text{th}}$ entry of $\boldsymbol{A}$ is given by:

$$A_{ij} = \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2$$

Then $\boldsymbol{A}$ is the Euclidean Distance Matrix for the data set $\{\boldsymbol{x}^{(i)}\}_{i=1}^{P}$.

Note that $\boldsymbol{A}$ is a symmetric matrix with 0's along its diagonal. Some properties of $\boldsymbol{A}$ are given in [64], and are summarized here:

Property 4.3.1 *If the data set X consists of different points, then $\boldsymbol{A}$ is nonsingular.*

Property 4.3.2 *If $v \in \mathbb{R}^P$ such that $\sum_{i=1}^{P} v_i = 0$, then*

$$v^{\mathrm{T}} \boldsymbol{A} v < 0$$

Let the matrix $\boldsymbol{\Phi}$ be defined as:

$$(\Phi)_{ij} = \exp\left(-A_{ij}^2\right)$$

then:

Property 4.3.3 *The matrix $\boldsymbol{\Phi}$ is positive definite.*

Definition 4.2 *The Transfer Matrix and Transfer Function*

Let the matrix $\boldsymbol{\Phi}$ be constructed by applying a function $\phi : \mathbb{R} \to \mathbb{R}$ to the euclidean distance matrix A elementwise. Then we will refer to $\boldsymbol{\Phi}$ as the transfer matrix associated with the transfer function ϕ.

Property 4.3.4 *For a large class of transfer functions, ϕ, the transfer matrix is nonsingular.*

For the proof and more on what the class of transfer functions is, see Micchelli [59].

There are many choices one can make for the transfer function. In theory, they are all equivalent in that they all form classes of basis functions that are dense in the space of continuous functions over compact domains. The choice is usually one of convenience, although the approximation can take fewer parameters if we choose the basis that will best represent the variation in the function. We return to this point after a few defining a few of the popular choices.

Common choices include:

$$\begin{array}{lcll}
\phi(r,\sigma) & = & \exp\left(\frac{-r^2}{\sigma^2}\right) & \text{Gaussian} \\
\phi(r) & = & r^3 & \text{Cubic} \\
\phi(r) & = & r^2\log(r) & \text{Thin Plate Spline} \\
\phi(r) & = & \frac{1}{r+1} & \text{Cauchy} \\
\phi(r,\beta) & = & \sqrt{r^2+\beta} & \text{Multiquadric} \\
\phi(r,\beta) & = & \frac{1}{\sqrt{r^2+\beta}} & \text{Inverse Multiquadric}
\end{array}$$

In choosing one set of basis functions over another, consider the nature of the data. For example, if the range has a wide variation, we may want to use the cubic over the Gaussian. on the other hand, if the data is smooth with very little variation, then the Gaussians will be a good choice. Thin plate splines and Multiquadrics have also been shown to be good routine choices [13].

4.3.1 Interpolation via Radial Basis Functions

Let $\{\boldsymbol{x}^{(i)}\}_{i=1}^P \subset \mathrm{I\!R}^N$. Suppose we also have a data set $\{\boldsymbol{y}^{(i)}\}_{i=1}^P \subset \mathrm{I\!R}^M$. Let $\boldsymbol{Y}$ be the P by M matrix constructed so that its i^{th} row is $(\boldsymbol{y}^{(i)})^{\mathrm{T}}$.

The interpolation problem is to determine a function $\boldsymbol{F}$ such that

$$\boldsymbol{y}^i = \boldsymbol{F}(\boldsymbol{x}^{(i)}),\ i = 1\ldots P$$

Assume $\boldsymbol{F}$ is defined so that the j^{th} element of $\boldsymbol{F}(\boldsymbol{x}^{(i)})$ is given by:

$$\boldsymbol{F}_j(\boldsymbol{x}^{(i)}) = \sum_{k=1}^{P} \omega_{kj}\ \phi\left(\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(k)}\|\right)$$

where the ω_{ij} are parameters to be determined. Then the interpolation problem translates to the following matrix equation, for which we need to solve for the constants $\omega_{ij} \in \Omega$, which is a P by M matrix.

$$\boldsymbol{\Phi}\Omega = \boldsymbol{Y}$$

By the properties listed above, this equation has a unique solution

$$\Omega = \boldsymbol{\Phi}^{-1}\boldsymbol{Y}$$

since $\boldsymbol{\Phi}$ is nonsingular, and ϕ has been chosen from the classes allowed.

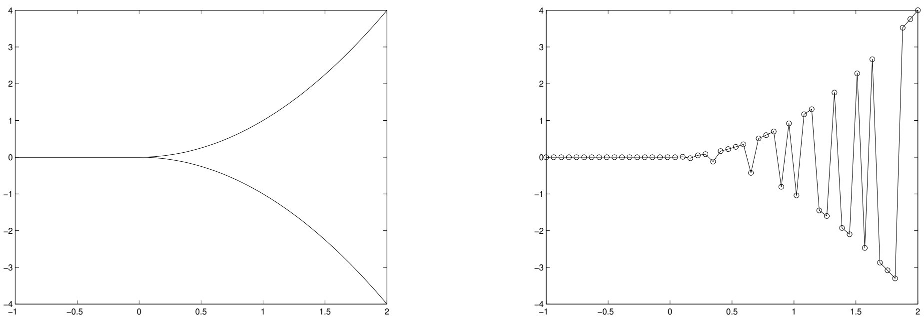

Figure 4.8: The image to the left is the original range set. The image to the right depicts the RBF solution to the discretized range. This shows that, while the RBF can obtain an exact solution to the interpolation problem, that solution is often not the optimal solution to generalize the range.

4.3.2 Generalization of Adaptive Networks and Radial Basis Functions

In most practical applications, it is *not* an interpolative solution that we are after. Rather, the data may be noisy, or there may not be enough data points to accurately reflect the function, and in general, an exact solution through many points may oscillate wildly between adjacent data points.

Consider the following example. The object in Figure 4.8 does not represent a manifold, and we cannot find a function from $\mathbb{R}^1$ to the object presented. However, as soon as we discretize the object to data points, then from the previous section, we can interpolate the data. One solution is presented in Figure 4.8. Note that the best (in terms of least squares error) one dimensional representation of this object should be the image of the zero function.

It is the goal of adaptive networks (including the three layer feedforward net of a previous section) to use the smallest number of basis functions possible to approximate the data as best as possible, and still be able to generalize the data. Therefore, there is a tradeoff between generalization to missing data and interpolation of the given data.

The paper of Broomhead and Lowe [13] discusses this extension of Radial Basis Functions, and we summarize their results here. The idea is that we replace the model of the function $\boldsymbol{F}$ so that:

$$\boldsymbol{F}_j(\boldsymbol{x}^{(i)}) = \sum_{k=1}^{K} \omega_{kj}\ \phi\left(\|\boldsymbol{x}^{(i)} - \boldsymbol{c}_k\|\right)$$

where we will refer to the set $\{\boldsymbol{c}_k\}_{k=1}^{K}$ as the set of Centers. Given a set of $K < P$ centers, we redefine the transfer matrix so that

$$(\boldsymbol{\Phi}_c)_{ij} = \phi\left(\|\boldsymbol{x}^{(i)} - \boldsymbol{c}_j\|\right)$$

and $\boldsymbol{\Phi}_c$ is now a P by K matrix. Once a set of centers has been chosen, then we have changed the interpolation problem to a problem of finding the least squares solution to:

$$\boldsymbol{\Phi}_c \Omega = \boldsymbol{Y}$$

whose solution is well known to be:

$$\Omega = \boldsymbol{\Phi}_c^{+} \boldsymbol{Y}$$

where $\boldsymbol{\Phi}_c^{+}$ is the Moore-Penrose generalized inverse of $\boldsymbol{\Phi}_c$, and

$$\boldsymbol{\Phi}_c^{+} \boldsymbol{\Phi}_c = I_{K \text{ x } K}$$

We will be using only this new form of the transfer matrix, and so will suppress the "c" subscripts for the remainder of this chapter.

Given the centers, the generalized inverse can be computed quickly from the reduced singular value decomposition of Φ:

$$\boldsymbol{\Phi} = USV^{\mathrm{T}}$$

where U is a P by K unitary matrix, S is a K by K diagonal matrix of singular values, and V is a K by K unitary matrix. The value of K is chosen so that there are no zero diagonal elements in S, so that S is easily inverted. Then:

$$\boldsymbol{\Phi}^{+} = VS^{-1}U^{\mathrm{T}}$$

with

$$S^{-1} = \mathrm{diag}\left(\frac{1}{S(1,1)}, \ldots, \frac{1}{S(K,K)}\right)$$

We are now in a position to re-examine the overfitting issue raised earlier. Suppose we take the centers to be some subset of the data (this is not a requirement, only a convenience). We will separate the data used in the previous example into two sets. The first part of the data will be used to determine solve the interpolation question, and the rest will be reserved to examine the ability of the network to generalize. This data set is referred to as the test set. Figure 4.9 gives us the tradeoff between generalization and interpolation very clearly. The set of centers was chosen randomly from the set of training data, and the number of centers increased from 1 to the entire data set (without replacement). The line punctuated with circles in the figure represents the mean square error (MSE) on the training set versus the number of centers. It is strictly decreasing to 0, as expected from our initial comments on the properties of the Euclidean Distance Matrix. On the other hand, the MSE on the test set is strictly increasing, again, as expected since the best fitting function we have in mind is the zero function. There are several techniques designed to optimize the trade off between accuracy and generalization, but they are outside the context of this brief overview. Rather, we discuss two signs of overfitting which we will utilize in building the RBF network.

In building an RBF network, we will try to use the minimal number of centers (which reduces the complexity of the model and does not allow for overfitting), and we will try to ensure that the coefficients of the RBF do not grow too large. In using this heuristic, we will utilize a measure of the complexity of the model by computing

$$\rho(N) = \frac{N_1}{N_2} \tag{4.7}$$

where N is the RBF model, N_1 is the number of centers taken from the data, and N_2 is the number of data points. Then we see that the maximum value of ρ is 1

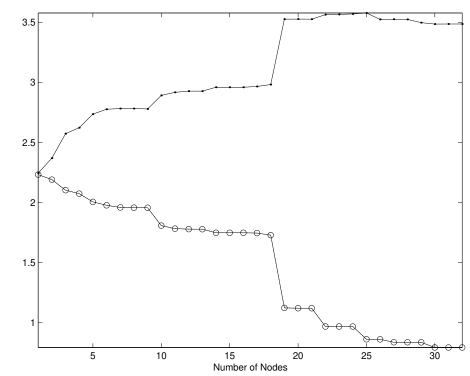

Figure 4.9: Mean Square Error on the training set (circles) versus the validation set (points). From this graph, the best solution is an RBF with only 1 center.

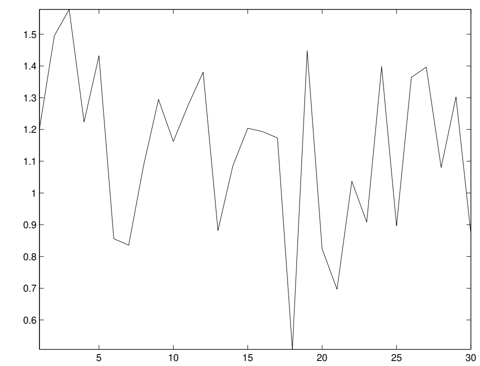

Figure 4.10: Mean Square Error using random sets of 20 centers for the interpolation matrix (30 trials total).

(and this implies that we have used all data for centers, and the minimum value is 0. We will try to keep ρ as low as possible while still attaining our desired accuracy. More details will be given in the examples of Chapter 6.

This example also illustrates one technique that has been often employed by users of RBF's: random center selection. The problem with this technique is that one may have the unfortunate circumstance of choosing a bad set of centers. Consider the following graph that depicts the MSE using 20 centers on the previous data set. The independent variable represents different random selections of the 20 centers.

The question we address now is whether there is an algorithm for determining a "good" set of centers.

Orthogonal Least Squares Solution for the Centers

The following is a summary of the work in the reference [14], where we will also follow that convention of assuming that the output is one dimensional. The multidimensional extension is straightforward, and is included in the Appendices, together with the Matlab implementation.

The idea is this: Suppose the interpolation matrix $\mathbf{\Phi}$ takes *all* data points for centers. This is clearly not optimal, for there should surely be some redundancy among the data, and we do not want a strict interpolation of the data, but good generalization. The goal then is to find those column vectors of $\mathbf{\Phi}$ which are most critical to the interpolation solution. Consider the column space of $\mathbf{\Phi}$ together with the range, $\boldsymbol{Y}$. We think of these columns as vectors, so that to determine which column of $\mathbf{\Phi}$ most closely matches the vector $\boldsymbol{Y}$, we compute the cosine of the angle between $\boldsymbol{Y}$ and the columns of $\boldsymbol{Phi}$. We see this computation in Equation (4.8), where it is called the error reduction ratio, but can be interpreted as:

$$\mathrm{err}_i = \cos^2(\theta)$$

for which we find the maximum. This occurs at $\theta = 0$ and $\theta = \pi$, and the minimum occurs at $\theta = \frac{\pi}{2}$. In choosing the maximum value, we choose the column that most closely points in the same direction as $\boldsymbol{Y}$.

The algorithm is an iterative procedure for orthogonalizing a subspace of the column space of $\mathbf{\Phi}$. As we select columns, we will remove them from $\mathbf{\Phi}$ and hold them in a matrix W. We will refer to $\mathbf{\Phi}$ by its columns,

$$\mathbf{\Phi} = [\Phi_1, \Phi_2 \ldots, \Phi_N]$$

Algorithm 4.2 *Orthogonal Least Squares for RBFs*

- For $1 \leq i \leq N$:
  - Initialize:
  $$\boldsymbol{w}_1^{(i)} = \Phi_i$$
  - Compute the error reduction ratio:
  $$\mathrm{err}_i = \frac{((\boldsymbol{w}_1^{(i)})^{\mathrm{T}}\boldsymbol{Y})^2}{||\boldsymbol{w}_1^{(i)}||^2||\boldsymbol{Y}||^2} \tag{4.8}$$
  In [14], it is shown that this value can be interpreted as the error associated with not using this column in the center selection process. The biggest value that err_i can take on is 1, which occurs if $\boldsymbol{Y}$ is orthogonal to $\boldsymbol{w}^{(i)}$.
  - Select the i that corresponds to the maximum error reduction:
  $$i = \mathrm{argmax}_j\{\mathrm{err}_j\}$$
- Main Loop on index k, with $1 \leq i \leq m$:
  - Compute the projection coefficients for $1 \leq j < k$:
  $$a_{jk}^{(i)} = \frac{\boldsymbol{w}_j^{\mathrm{T}}\Phi_i}{||\boldsymbol{w}_j^{(i)}||^2}$$
  - Deflate the column space of $\boldsymbol{\Phi}$:
  $$\boldsymbol{w}_k^{(i)} = \Phi_i - \sum_{j=1}^{k-1} a_{jk}^{(i)}\boldsymbol{w}_j$$
  - Compute the error:
  $$\mathrm{err}_k^{(i)} = \frac{((\boldsymbol{w}_k^{(i)})^{\mathrm{T}}\boldsymbol{Y})^2}{||\boldsymbol{w}_k^{(i)}||^2||\boldsymbol{Y}||^2}$$
  - Find the value of $\mathrm{err}_k^{(i)}$ which is maximal. Put the corresponding column into W, and deflate $\boldsymbol{\Phi}$ by the corresponding column.

- Stopping Criterion. We can use one of several options:
  - Suggested by [14]: Terminate at the k^{th} step, where

    $$1 - \sum_{j=1}^{k} \mathrm{err}_j < \rho$$

    for a predetermined ρ.
  - Stop when the error on the residual is below a threshold.
  - Stop when the next chosen vector, ϕ is of small enough size.

The orthogonal least squares solution provides an efficient method of center selection, avoiding the problems of perhaps choosing a bad set at random. The problem with this algorithm is in the computation of the matrix $\mathbf{\Phi}$. Note that the size can become very large quite quickly.

On the other hand, notice that the RBF interpolation problem suppresses the curse of dimensionality (that one finds in other methods) quite handily. The only real restriction to solving this interpolation problem is in the number of data points that one can input and still solve for the inverse (or pseudoinverse) in a reasonable amount of time.

4.4 Conclusions

We have presented several neural networks in this chapter. The main use we have focused on is the neural network's ability to interpolate and generalize input-output pairs, at the same time minimizing the curse of dimensionality that other interpolation methods cannot.

We have also presented an application of a novel computational node, the Spherical Node, and have shown the usefulness of mixing techniques, such as Kohonen's Map, to give a triangulation of complex data.

At the beginning of the research into methods suitable for implementation in the neural charts algorithm, we focused on the multilayer feed forward network and the autoassociation network. While these networks were successful (they are used in some of the examples later on), the nonlinear optimization of the weights is much more computationally expensive than the radial basis function training, as the data sets we worked with were relatively small.

The computational expense for training feed forward neural networks grows quickly with the number of hidden nodes. For that reason, they are useful more for smooth data than for more complex models. On the other hand, the additional computational expense for adding more training points is quite less than for the radial basis networks. The radial basis network is better for smaller data sets — the extra expense for more centers is significantly less than that for the feed forward network.

For the neural charts, we do not deal with extensive amounts of data in each cluster, therefore we have found that the radial basis networks can be much quicker to implement than the nonlinear optimization associated with the feed forward net.

In the next chapter, we will discuss the role of the neural network in the neural charts algorithm in detail.

Chapter 5

THE NEURAL CHARTS ALGORITHM

In Chapter 1, we gave a general outline of the neural charts algorithm. Here we give the specific details of the implementation. We discuss two versions of the algorithm, each gives useful information about the data that is presented. The second version is the current one, but we will also discuss the preliminary algorithm in order to discover the shortcomings, insight and motivation in constructing the current version.

5.1 The Problem and Form of Solution

We are given a set of data that represents the long term dynamics of a certain smooth dynamical system. The data is high dimensional, but it is suspected that the long term dynamics reside on a low dimensional set. The problem is to reduce the dimensionality of the dynamical system. The neural charts algorithm will accomplish this reduction via local linear projections to the local tangent planes followed by constructing nonlinear local reconstructions of the tangent planes back to the manifold in which the long term dynamics reside. The output of the neural charts algorithm:

1. A set of centers and a decision function. The domain of the decision function is a data point from the local manifold, and the output of the decision function is the index to the cluster for which the data point belongs.

2. Associated to each cluster, a set of basis vectors is stored that represent the span of the local tangent space. The local dimension is also stored, and is the number of basis vectors. These vectors also represent the projection matrices, π_i, introduced in Chapter 1.

3. Associated to each cluster is a neural network. The domain of the network is the local tangent space, and the image of the network is the local manifold.

With these items, it is possible to completely analyze the structure, dimension and topology of the local manifold that contains the attracting set to the dynamical system. We also see here that we have a local numerical version of an "atlas of charts" — hence the name. From these charts, we can represent the equations of motion using the smallest possible number of degrees of freedom, and track how the flow is progressing across the manifold. These applications are detailed below.

5.2 Preparation of the Data

If we suspect that there is a lot of redundancy in the current representation of the data, then a global Karhunen-Loève expansion should be performed to remove those redundancies, filter excessive noise, and prepare the data for local analysis. We remark here that if there is excessive amounts of noise in the data, then a local analysis is not the approach one should take, as each data point carries much more weight when analyzing locally than when analyzing the data globally.

5.3 Clustering of the Data

Clustering of the data has been implemented via the use of a tree-structured algorithm. The clustering tree makes for swift location of cluster centers, and is adaptive enough for the algorithm to be adaptive in cluster placement. The general tree algorithm is independent of the clustering algorithm used. We summarize it below, giving some specifics for each type of clustering.

Algorithm 5.1 *Tree Structured Clustering*

Begin the algorithm by setting the Root Node of the Tree to the mean of the data.

1. Split the cluster.

   (a) Kohonen's Map:

   Obtain an estimate of the local dimensionality, k, of the parent node. Use $2k$ vectors for the index set. Let $\{\boldsymbol{v}_i\}_{i=1}^{k}$ be the k eigenvectors of the KL expansion of the data set. Then define the index vectors:

   $$\{\boldsymbol{c}_{i_1}, \boldsymbol{c}_{i_2}\} = \pm \boldsymbol{v}_i$$

   The tree in this case has a variable number of child nodes for each parent, and each node spawns $2k$ children at each step.

   (b) Neural Gas and Space-Time LBG:

   Given the parent node is $\boldsymbol{c}$, initialize two child nodes as:

   $$\{\boldsymbol{c}_1, \boldsymbol{c}_2\} = \boldsymbol{c} + \nu_{1,2}$$

   where ν is a small, random vector. This is therefore a binary tree — much easier (and more standardized) to code.

2. Run the clustering algorithm until the stopping criterion has been observed. See the individual sections for the stopping criteria.

3. Decide if each cluster needs to be split. Decide not to split if:

   (a) Kohonen's Map:

   A subsequent splitting of the cluster results in subclusters all of the same dimension as the parent. This means that a subclustering will not result in further dimensionality reduction.

(b) Neural Gas and Space-Time LBG:

The distortion errors (defined in each of the individual sections) have attained a preset level.

In both cases, we should also stop subdividing if the number of points in the cluster starts to get too small. For the KL expansion in the next step, we should retain the number of points equal to least twice the dimension of the space.

4. Continue from Step 1 until all Nodes are finished splitting.

For the previous step of the algorithm, all coding was done in C. Matlab was very slow in performing the clustering, since there are many loops that cannot be vectorized. For the Kohonen's Map and Neural Gas versions of the algorithm, all steps of the neural charts algorithm were coded in C. For the Space-Time algorithm, at this stage, the C program outputs a tree structure that is suitable for reading into Matlab (only v.5 and higher versions, since we are defining our own data types).

5.4 Building the Charts

For the earliest versions of the neural charts code that used Kohonens SOM and the Neural Gas algorithm, the local dimension of the tangent space was found via Fukunaga's ad-hoc cutoff. For the version utilizing the Space-Time LBG, the Modified Local SVD approach was used.

Suppose that the local data set is given as a matrix X, where the i^{th} row corresponds to the i^{th} data point, so that X is P_j by N. Let U be the matrix of K_j eigenvectors of the covariance matrix of X where K_j is the local dimension of the tangent space. Then the span of these eigenvectors approximates the span of the tangent space (see Appendix C), and we define the projection and linear reconstruction functions setwise on the matrix X by:

$$\pi_j(X) = XU \quad \text{The Projection Function} \tag{5.1}$$

and

$$\tilde{\pi}_j(X) = XUU^{\mathrm{T}} \quad \text{The Linear Reconstruction} \tag{5.2}$$

so that the error on the reconstruction of the matrix X is given by:

$$X(UU^{\mathrm{T}} - I)$$

The dimensionality algorithms were given the covariance matrix of the local clustered data, and each output the basis vectors and local dimension. In the C code, these were stored in the node data structure, and in the Matlab code, a new tree was formed that contained this data.

5.5 Building the Nonlinear Inverses

The following section gives the details that were used in building the neural inverses to the projection matrices above.

Key to the success of the nonlinear reconstruction is the requirement that the transition from one reconstructed cluster to the next is as smooth as possible. If we were to train the networks simply on the projected cluster, then we run the risk of having non-smooth transitions. To counteract this effect of interpolation, where we may have large errors on the edges of the data, we increase the size of each cluster slightly for training. This is accomplished by exploiting the fact that the data is *time ordered* as a trajectory in $\mathbb{R}^N$. The data in each cluster is therefore time indexed, so that if $\boldsymbol{x}^{(i)}$ is in a cluster, then its value at the next time value is simply given by

$$\boldsymbol{x}^{(i)} \rightarrow \boldsymbol{x}^{(i+1)}$$

Algorithm 5.2 *Constructing the Domain of the Neural Chart*

We increase the cluster by adding a set number of time steps to each trajectory through the cluster. Therefore, the input to this algorithm is the local data with the original indices, and the number of time steps, k, to add. Let L be an integer vector of length P_j (for the j^{th} cluster) so that its i^{th} component is the index for data point i.

For $i = 1, 2, \ldots, P_j$:

For $m = 1, 2, \ldots k$:

Search the first P_j components of L for $L(i) + m$. If it is not there, increase the vector L so that its last element is $L(i) + m$.

End of both loops.

Next, set $\tilde{X} = \pi_j(X)$, where X is the extended local data whose original indices are listed in the vector L constructed above.

Now we construct the range values for the Neural Networks. Note that since we already have an approximate value for the reconstructed manifold, then it would be redundant for the network to learn this piece of the reconstruction. Therefore, we let the networks learn the *error* between the original data and the linearly reconstructed data. In effect, the neural network is mapping:

$$\psi_j : \tilde{X} \to X(UU^{\mathrm{T}} - I) \text{ or } \tilde{X} - X$$

Pointwise, the neural network is used for the following approximation:

$$\boldsymbol{x}^{(i)} \approx \tilde{\pi}_j(\boldsymbol{x}^{(i)}) + \psi_j(\boldsymbol{x}^{(i)})$$

Training now proceeds. A key feature to this algorithm is that, if the network fails to train to a given tolerance, then that is indicative of an error in the selection of the local dimension. The local dimension must be large enough so that an accurate reconstruction can be made. This also gives a check to see that the local dimension was accurate enough.

Algorithm 5.3 *Training of the Neural Networks*

1. Select the size of the network.

   (a) Feedforward Network: Set the size of the hidden layer to a constant multiple of the local dimension. This was a problem-dependent parameter. In the next chapter, this number of hidden nodes was set at three times the local dimension.

   (b) Radial Basis Functions, Random Center Selection: Begin with one center.

   (c) Radial Basis Functions, using Orthogonal Least Squares. The algorithm sets the number of centers.

2. Train the network. If it fails to train to the specified error tolerance, then:

   (a) Feed Forward Network: Increase the number of hidden nodes, by multiples of the given constant. If the network continues to fail to train (at most twice), increase the dimensionality and begin again until training succeeds.

   (b) Radial Basis Functions, Random Center Selection: Continue to select centers until either the training criterion is met, or until 80% of the data has been used as centers. If 80% of the data has been used, and the error is not below tolerance, then increase the dimensionality of the data and begin again. Continue until training has succeeded.

   (c) Radial Basis Functions, Orthogonal Least Squares: If the number of centers chosen is less than 80% increase the number of centers to that point. If training has still not been successful, increase the local dimension until training is successful.

5.6 Local Integration using Neural Charts

The following method, reiterated from Chapter 1, highlights one use of the neural charts algorithm: Local Integration. We repeat the algorithm here:

Algorithm 5.4 *The Local Integration Algorithm*

1. Choose an initial $\mathbf{x}$.
2. Find the local chart for $\mathbf{x}$.
3. Project $\mathbf{x} \in U_i$ to $\mathbf{v} \in V_i$.
4. Perform one integration step on $\mathbf{v}$ (using equation (1.3) to find dv values) to obtain the new local value $\mathbf{v}$.
5. Perform neural reconstruction to $\mathbf{v}$, obtaining a new $\mathbf{x}$.
6. Repeat from step (2).

From this algorithm, it it clear that any integrator used will be searching the *tangent space* for the appropriate path. In the special case of a cycle, the tangent space is one-dimensional, so the appropriate path is found by simply adding (or subtracting) a small increment from the current value. The direction of the original flow across these domains will need to be ascertained prior to integration, to determine whether to add an increment, h, or subtract it. In either case, step (4) above can be replaced by:

4. Increment v to $v \pm h$ (v is 1-dimensional)

The results of this integration technique are shown using the reduced Kuramoto-Sivashinsky equation in the examples.

Consider also that the local dynamics are given by:

$$\dot{\boldsymbol{v}} = \pi_i F(\psi_i(\boldsymbol{x}(t))$$

and an Euler approximation to the solution, with variable time step $h_t(v)$ is

$$\boldsymbol{v}(t + h_t) = \boldsymbol{v}(t) + h_t(v) \cdot \dot{\boldsymbol{v}}$$

with $\psi_i(\boldsymbol{v}) = \boldsymbol{x}$ being the neural reconstructions of the local manifolds. Then with only a slight modification, we can construct local models of the flow by changing the range set of ψ_i so that

$$\hat{\psi}_i(\boldsymbol{v}(t)) = \boldsymbol{x}(t + h_t) \tag{5.3}$$

We summarize this restructuring of the neural networks in the following:

Algorithm 5.5 *The Neural Model of the Flow*

1. Begin with a given $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$.
2. Locate the cluster to which this point belongs, and project to obtain $\boldsymbol{v}(t)$.
3. Use the neural model of the flow to advance the solution one step forward in time. See Equation (5.3).
4. From the previous step, compute $\dot{\boldsymbol{x}}$.
5. Repeat from Step 2.

The image in Figure (5.1) is an example taken from one of the clusters in the "thin torus" in Chapter 6. The flow is designated by the straight line- the initial point is an asterisk, and the trajectory one unit in time forward is represented by a circle. The triangles give the neural approximation to the set one unit in time forward (so they are approximating the data represented by the circles).

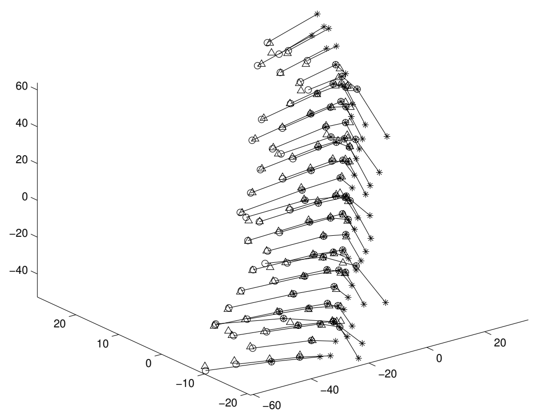

Figure 5.1: A sample of the neural model of the flow. The original flow is designated by a line, the asterisk is the initial point and the circle is the endpoint. The triangles are the neural approximation to the circles. The data set was rescaled for better visualization.

Thus, we build an additional neural network at each terminal node of the clustering tree that interpolates the varying time steps inside that cluster. The success of this algorithm depends on how well the initial clusters were able to incorporate the temporal derivative information. Since the Space-Time LBG algorithm is designed to reduce the cluster to a dynamically homogenous regime, the neural model of the flow across each cluster should be relatively simple (in terms of the complexity of the model).

In Chapter 6, we utilize RBF networks to accomplish the neural integration, and find that we are able to avoid the time consuming standard integration (especially with the Kuramoto-Sivashinsky Equation), where the neural model of the flow can be computed at much less computational expense.

5.7 Summary and Conclusions

The neural charts algorithm is a novel technique that provides a local semi-analytic description of the manifold on which the long term dynamics of the system reside, and also provides a basis on which we can construct local equations of motion from the data.

There are several innovations taking place in the neural charts algorithm. A summary of the highlights are given below:

1. The Space-Time LBG: No clustering algorithm had been set up to specifically deal with the clustering of data from an attracting set to a dynamical system. The Space-Time algorithm represents an innovation that can be extended to other clustering algorithms (here we modified the LBG algorithm, but a similar metric can be used with Kohonen's Map and the Neural Gas algorithms). Being the first step in most local modeling tasks, this is a key element.

2. The Modified Local SVD method for determining the local dimension of the tangent space also represents an innovative change to current methods, which required ad-hoc cutoffs. The modifications, together with the Space-Time clustering algorithm, represent the best method yet to automating the determination of the local tangent space dimension. Back in the $70's$, Fukunaga commented that his algorithm required a clustering technique to determine dimensionality. His current algorithm requires "expert assistance" to determine appropriate cluster size and placement.

3. The use of Neural inverses. We have used neural networks in a novel way- to determine the nonlinear inverse to the Karhunen-Loève truncation. This approach allows us to use less clusters of bigger size than the techniques described in Chapter 7. The clusters we use are also more natural, with each describing regions where both space and temporal derivatives are approximately equal throughout the data.

4. We have performed a local neural approximation to the (local) equations of motion from the data.

5. We have extended the neural charts algorithm in the last example of Chapter 6 by incorporating knowledge of the dynamics in a global model. The initial decomposition reduced the dynamics to a left travelling wave, a right travelling wave, and an excursionary regime. We are now able to further decompose the two stable regimes using the neural charts algorithm from which we show that the interplay between the global and local models are able to successfully reproduce the original dynamics.

The output of the neural charts algorithm provides for a jumping off point for other tasks. For example, we can now obtain local equations of motion quite simply. For each cluster, use a neural network to obtain the function that carries each point in that cluster to its next point in time. This technique of time series prediction was also used by Broomhead and Lowe ([13]) and Moody and Darken ([60]), although it was done on a global scale. It might prove to be more beneficial to learn how the complexity of the flow changes over the manifold — which is something that can be analyzed given its local description. The local description of the manifold can also be used to inspect the local direction fields, help in the computation of Lyapunov exponents, and more generally, help in any computation that is currently done on a global scale. There is much in the way of future research that can be done with the local description that we can now compute.

Chapter 6

EXAMPLES

Now we illustrate the neural charts technique by means of several examples. We incorporate knowledge of the dynamical system into the neural charts model in several places. If the topology is known, we model the topology using Kohonen's Map. If unknown, we use either Neural Gas or the Space-Time LBG. In the first four examples, we use Fukunaga's energy cutoff for the evaluation of dimensionality, and use the Modified LSVD algorithm for the last two examples. Also in the first examples, we use a feedforward network for the manifold reconstruction process, and in the last two examples, we use RBF networks. In the last example, we incorporate a global model over the top of the neural charts model in order to gain a clearer understanding of the complex dynamics presented. Additionally, in the last two examples, we present the local neural equations of motion, described in Chapter 5. We show that with only a slight modification to the training sets, the neural networks (in these cases, the RBF networks) are able to accurately reproduce the flow. In the last example, the interaction between the global and local models provides us with additional insight to the complexity of the dynamical system.

In all, we show the adaptability of the neural charts algorithm in incorporating prior knowledge to get the best representation possible for the model of the manifold, and in assisting the local analysis and interpretability of the flow.

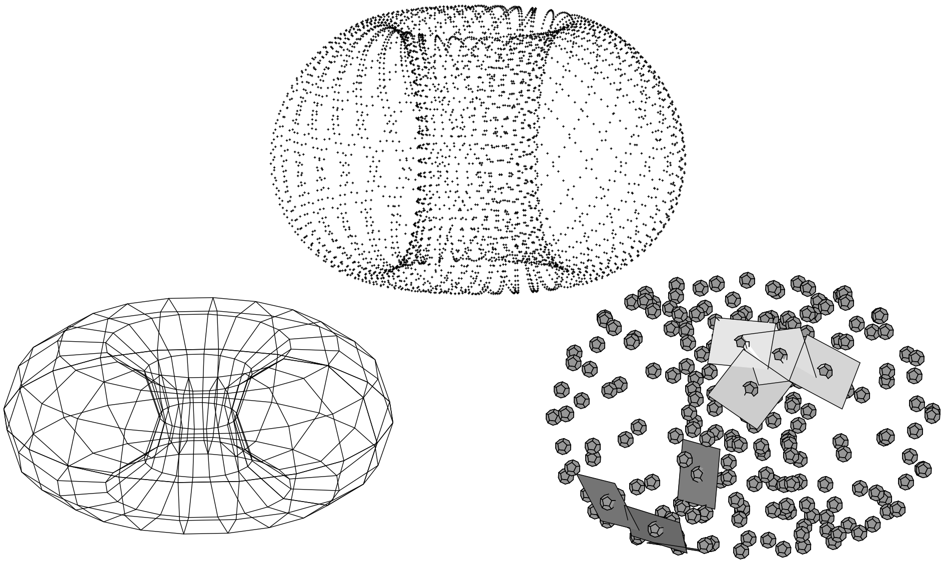

Figure 6.1: Top: Original data set. Left: The index set for the SOM, chosen to reflect the topology of the torus. Right: Cluster centers and some tangent planes.

6.1 The Torus

The following is an illustrative example of the neural charts algorithm using the SOM as the basis for clustering. Consider the system of equations

$$\begin{aligned}
\dot{x_1} &= (\lambda - b)x_1 - cx_2 + x_1\,(x_3 + d(1.0 - x_3^2)) \\
\dot{x_2} &= (\lambda - b)x_2 + cx_1 + x_2\,(x_3 + d(1.0 - x_3^2)) \\
\dot{x_3} &= \lambda x_3 - (x_1^2 + x_2^2 + x_3^2)
\end{aligned} \tag{6.1}$$

with $\lambda = 2.02$, b=3.0, c=0.25, d=0.02, and initial condition (0.1, 0.1, 0.1) as suggested in [32]. Then the Ω-limit set is a torus. The data was produced using a Runge-Kutta integration routine with time step 0.01 (see the upper picture of Figure 6.1). A set of 200 clusters was used and was enough to resolve the toral shape, i.e., if we consider the torus as $S^1 \,\mathrm{x}\, S^1$, then the 200 points in the index set were distributed as $20 \,\mathrm{x}\, 10$, as seen in the bottom left picture of Figure 6.1.

The outcome of the SOM is also shown (bottom right picture in Figure 6.1). We then locally decomposed the torus into regions using (2.3), and performed

Table 6.1: Sample Eigenvalues: Torus

Cluster	Dim 1	Dim 2	Dim 3
11	0.602	0.395	0.0016
100	0.521	0.476	0.0024
180	0.551	0.445	0.0025

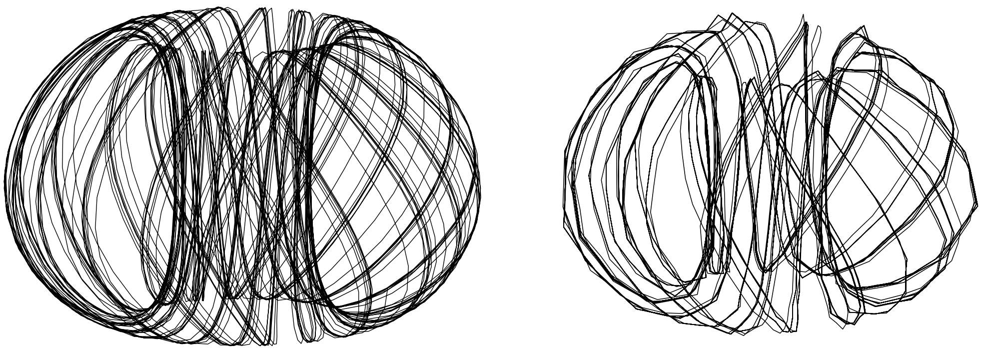

Figure 6.2: Left: Neural Reconstruction. Right: Linear Reconstruction

local KL to each, resulting in approximations to the local tangent planes (two dimensional). Several of these planes are shown in the bottom right picture of Figure 6.1, and a few sample eigenvalues (relative energy, equation (3.3)) are shown in Table 6.1. The table clearly shows the division between the two dimensional tangent space and the three dimensional space in the eigenvalue distribution. Each of the neural inverses were then constructed using feedforward, three layer (input, hidden, output) neural networks with 2-12-3 nodes (2 nodes in the input layer, 12 nodes in the hidden layer, and 3 nodes in the output layer). The result of local integration is shown in Figure 6.2, where the transitions between local patches is quite smooth. This is compared with integration using the linear reconstruction (i.e., replace the step with using ψ_i for reconstruction with $\tilde{\pi}_i$ (also see equation (5.2)).

As is seen in the picture, the linear dynamical reconstruction introduces jumps to the flow (between tangent planes), and is thus less desireable than the smooth neural dynamical reconstruction. In fact, in the linear reconstruction, it required several attempts to find a step size for the solution that didn't either converge to a periodic cycle or become "locked" between tangent planes, behavior that was not warranted by the initial condition. The next example actually combines some neural reconstructions with linear reconstructions.

6.2 The Lorenz Attractor

Consider the following system of equations, solutions of which result in obtaining an approximation to the "Lorenz Butterfly":

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= rx - y - xz \\ \dot{z} &= xy - bz \end{aligned} \tag{6.2}$$

with $\sigma = 10$, $b = \frac{8}{3}$, and $r = 28$. We include this attractor for its change of local dimensionality and low dimensional dynamics which are easy to depict graphically. The correlation dimension of this fractal attractor has been computed to be approximately 2.05 (for more information, see [70], pg. 421 and the references contained therein), which suggests that although the attractor is almost planar in the two leaves, there exist local regions which cannot be embedded into less than three dimensional space. Here the Neural Gas technique is used to show how the algorithm attempts to uncover the connectivity structure of the attractor. The first picture (Figure 6.3) shows the result of applying the Neural Gas algorithm to 100 cluster centers. A large number of cluster centers in this example was used to show how the algorithm resolves the connectivity of the Butterfly; the neural charts algorithm itself does not require this large number. As depicted in Figure 6.3, the Neural Gas algorithm has been successful at uncovering the two holes that

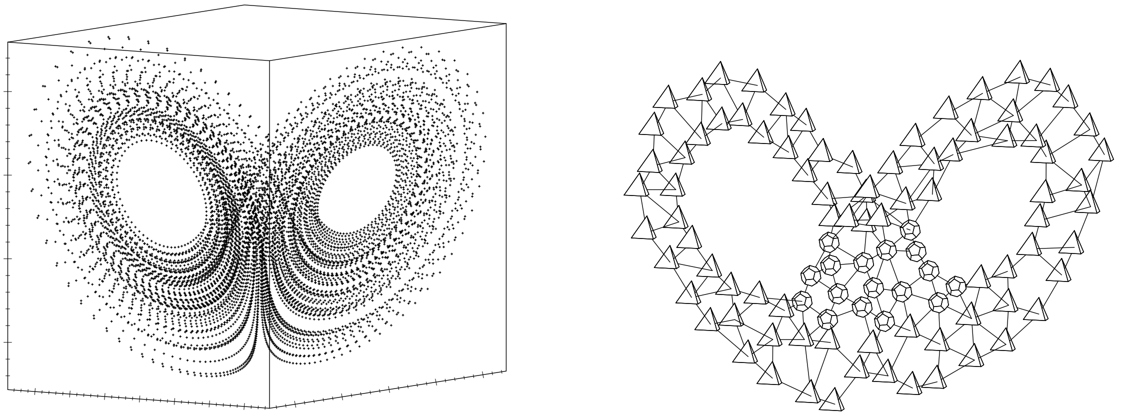

Figure 6.3: Left: Original data. Right: Cluster centers and spatial connections. Two dimensional centers are shown as tetrahedra and three dimensional centers are shown as dodecahedra

appeared in the original data set (the original data set was chosen to accentuate these features for demonstration purposes; see the training data in Figure 6.3). In the same picture, cluster centers deemed two dimensional by the KL procedure are shown as tetraheda, and centers deemed three dimensional are shown as dodecahedra. As we know, the "trouble" region for decomposing the Lorenz Attractor is the region in which the two Cantor-like leaves conjoin, resulting in a subset of the data that cannot be embedded in less than three dimensional space. Table 6.2 shows a sampling of relative energy eigenvalues. The first and last entries correspond to two dimensional regions, and the middle entries correspond to three dimensional regions. Cluster 9 represents a borderline case, and in such cases, the algorithm should ensure a high enough dimensionality so that the projection map to the tangent space is $1-1$, thus this cluster was classified as three dimensional. The regions classified as two dimensional had a neural inverse with a 2-4-3 architecture. The three dimensional regions, of course, did not require a neural inverse, so one might think of these regions being linear reconstructions; when no compression is possible, a full set of eigenvectors is used.

The picture to the left in Figure 6.4 is of the neural reconstruction of the training data, and the transitions between two and three dimensional regions are

Table 6.2: Sample Eigenvalues: Lorenz

Cluster	Dim 1	Dim 2	Dim 3
6	0.817	0.182	0.000072
9	0.561	0.338	0.099
56	0.475	0.311	0.213
78	0.588	0.411	0.000676

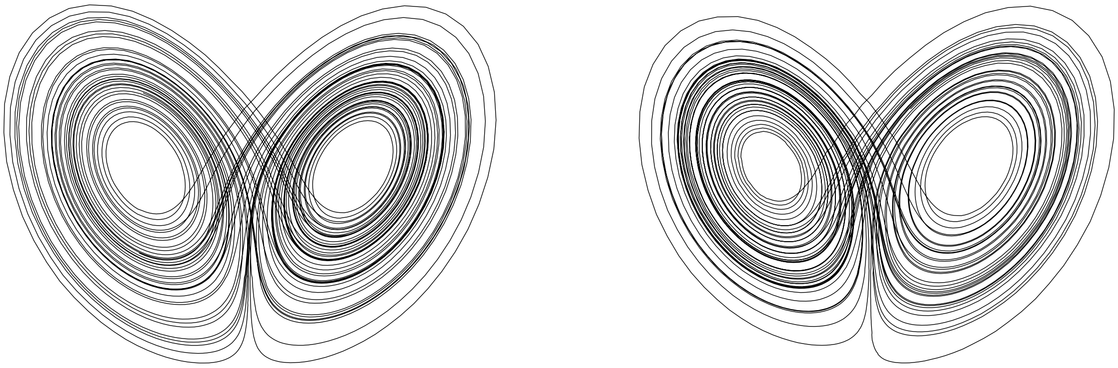

Figure 6.4: Left: Neural Reconstruction. Right: Linear Reconstruction. In this case the reconstructions are almost identical, showing that the attractor is very close to being locally planar.

smooth. The picture next to it is the linear reconstruction using the local integration algorithm. This time, the linear model has shown itself to be good at a close approximation for the two leaves of the attractor.

In this example, the utility of the clustering algorithm in discovering the topological structure of the manifold has been demonstrated, and has provided us with an ability to analyze how the flow moves between clusters and changes dimensionality. In the next examples, we will use fewer clusters to illustrate other features of the neural charts algorithm.

6.3 The Kuramoto-Sivashinsky (KS) Equation

The data used in the examples represent approximations to $u(x,t)$, solutions to:

$$u_t + 4u_{xxxx} + \alpha\left(u_{xx} + \frac{1}{2}(u_x)^2\right) = 0 \tag{6.3}$$

with $(x,t) \in \mathbb{R}\text{x}\mathbb{R}_+$ and periodic boundary conditions, $u(x,t) = u(x+2\pi, t)$.

The KS equation models pattern formations in different physical contexts and is a paradigm of low dimensional behavior in solutions to partial differential equations [40, 41].

It was shown that solutions to the KS equation are equivalent to solutions from a finite dimensional system of ordinary differental equations, in that solutions to the PDE are being exponentially attracted in an "inertial manifold" [15]. Also, low dimensional approximations to the inertial manifold have been constructed for the KS equation in our context, with periodic boundary conditions [17].

This equation is interesting for the range of behaviors that can be found by adjusting the bifurcation parameter, α, and for the sensitivity of solutions to the integration error. In [40, 41], a cautionary note suggests that local truncation error of less than 10^{-6} may result in the loss of certain solution behavior, and that small effects on that order in some Fourier modes critically impact the dynamics.

As in [47], we summarize the construction of the finite dimensional system of ODEs via the Fourier-Galerkin method.

Let

$$u(x,t) = \sum_{n=-\infty}^{\infty} a_n(t)\mathrm{e}^{inx}$$

Then, substituting this *ansatz* into Equation (6.3) and using the orthogonality relationships for e^{inx}, the expansion decouples a_0, and with $a_n = \bar{a}_{-n}$, we obtain the following $2N$ system of equations:

$$\dot{a}_l = l^2(\alpha - 4l^2)a_l + \frac{\alpha}{2}\sum_{n=1}^{l-1}(l-n)na_{l-n}a_n - \alpha\sum_{n=1}^{N-1}(l+n)na_{l+n}\bar{a}_n$$

for $2 \leq l \leq N-1$, and for $l = 1$ and $l = N$:

$$\dot{a}_1 = (\alpha - 4)a_1 - \alpha \sum_{n=2}^{N} (1-n)n\bar{a}_{n-1}a_n$$

and

$$\dot{a}_N = N^2(\alpha - 4N^2)a_N + \frac{\alpha}{2} \sum_{n=1}^{N} (N-n)n\bar{a}_{N-n}a_n$$

We used a 10 complex mode expansion. As discussed in [17, 1], the periodic boundary conditions introduces an action of $O(2)$ on the Fourier-Galerkin modes and also on the set of ODEs gained from the projection. Therefore, each of the steady state solutions is actually a whole circle worth of steady state solutions which one finds by acting on one representative of the symmetry group. The examples we present are thus numerical approximations to the forward attracting set of one of these representations.

We use an Adams-Bashforth predictor-corrector integration scheme with adaptive stepsize, where we take care to produce estimates of local truncation error to between 10^{-6} and 10^{-10}.

We use a Galerkin procedure to project the solution to Equation (6.3) into 10 complex Fourier modes (therefore, our data is in $\mathbb{R}^{20}$). It has been shown (see, for example, [47] and [46]) that this is a sufficiently high dimension to resolve the attractors present here. Since there will be multiple representations of the solution space, let us agree to call the original data space (with Fourier coefficients in $\mathbb{R}^{20}$) "Fourier space", the solutions depicted by the real-valued $u(x,t)$ "physical space', and the local tangent space will be referred to as "local space".

The Dynamics of the KS Equation, $\alpha = 52$ to 89

We focus on this parameter regime because of the low values of α which will lend themselves to 20 dimensional simulation, and because, as stated in [40, 41], the route to chaos we describe here is generic in the sense that the same routes occur for higher parameter regimes such as $\alpha = 117.5, 187$, and 253 [40, 41]. Beginning

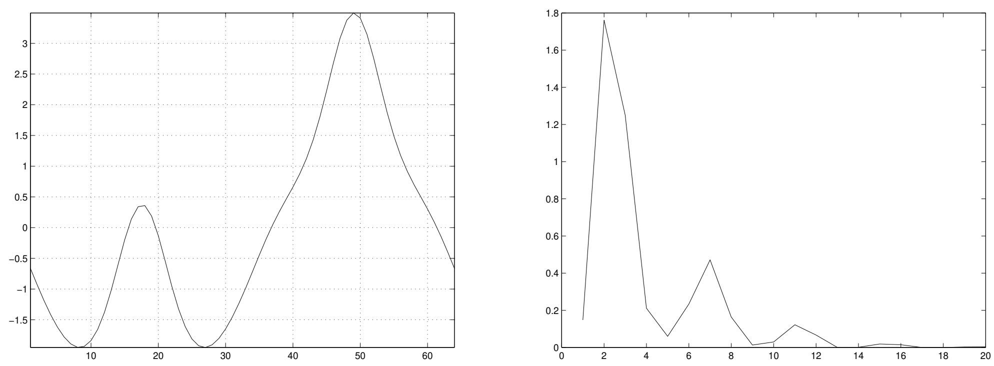

Figure 6.5: The strange fixed point at $\alpha = 72$ and it's Fourier power spectrum. This figure will play a dominant role in understanding the flow at $\alpha = 68$.

at $\alpha = 52$, the attracting set for the KS equation is a locked travelling wave. There is a multiple Hopf bifurcation at $\alpha = 66.751$ at which there are 4 eigenvalues on the imaginary axis [1]. There are three branches of periodic solutions emanating from this bifurcation, precisely one of which is stable. At 67.323, this branch loses its stability. At this point, we begin obtaining bursting behaviors from the periodic solution to the trimodal fixed point and towards the strange fixed point at $\alpha = 72.0$ [47].

At $\alpha = 72$, we obtain a global attracting point (See Figure 6.5). The dynamics have undergone a reverse bifurcation back to stability [40, 41]. It persists as a global sink until a Hopf bifurcation occurs at $\alpha = 83.7241$ [1]. At $\alpha = 84.25$, we obtain rapid localized oscillations, which are the forerunner of spatially concentrated zones of turbulence [40, 41]. At $\alpha = 85.672$, there is secondary bifurcation to a travelling beating wave [1]. We analyze this behavior in depth at $\alpha = 87$. At $\alpha = 89$, we return to a chaotic state. The dynamics are summarized in Table 6.3.

We examine the forward attracting sets in detail for a few of these parameter values.

α	Behavior
52*	Locked Travelling Wave
66	Spiral Attracting to Trimodel State
66.751	Multiple Hopf Bifurcation
67	Beating Wave
68*	Complex oscillatory and chaotic motion
72	Strange Fixed Point
83.7241	Hopf Bifurcation
84.25*	Beating Wave
85.672	Secondary Bifurcation to a travelling beating wave
87*	Travelling Beating Wave, a “thin torus”.
89	Chaotic State

Table 6.3: Description of Selected values of α. The values with an asterisk are presented in detail in the examples.

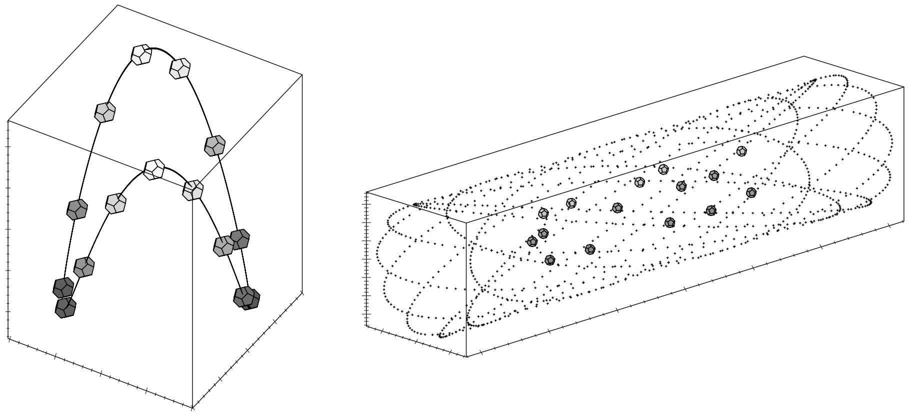

Figure 6.6: Fourier projections for $\alpha = 52.0$. Left: First three modes. Right: Last three modes

Table 6.4: Sample Eigenvalues: $\alpha = 52.0$

Cluster	Dim 1	Dim 2	Dim 3
5	0.979	0.020	0.0001
10	0.980	0.019	0.0001
180	0.979	0.020	0.0001

One Dimensional Trajectories: $\alpha = 52.0$ **and** $\alpha = 84.25$

We begin with two simple examples in that the neural charts algorithm uncovers the local dimensionality to be one, implying that the attractors are knots in Fourier space, and armed with this knowledge, one can perform a global reduction of the space to S^1.

Example: $\alpha = 52.0$

In Figure 6.6 we show the results of projecting the original trajectory we obtained numerically into two, three dimensional cross sections. The first picture is the projection into the first three dimensions of Fourier space, and the second picture is for the last three (dimensions 18, 19 and 20). Also shown are the corresponding projections of the 15 cluster centers used to decompose the manifold. For this value of α, the attractor corresponds to a locked travelling wave, which is

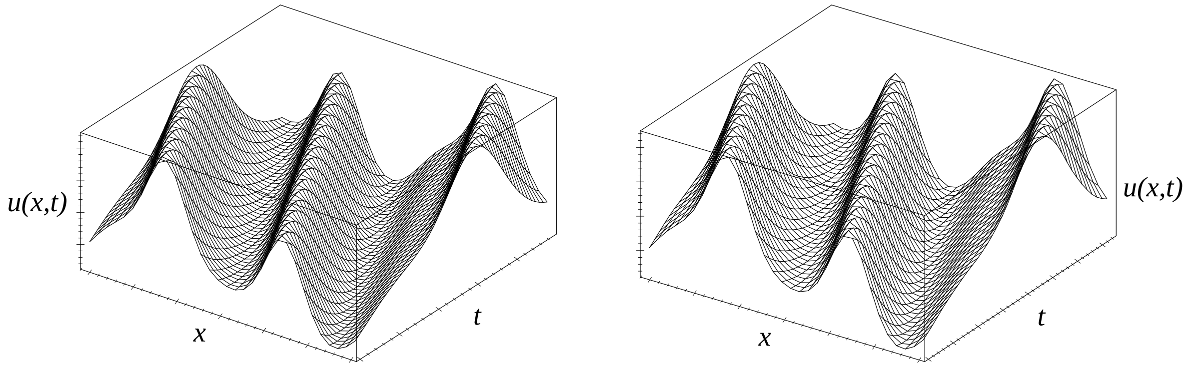

Figure 6.7: Physical space for $\alpha = 52.0$. Left: Original Data set. Right: Neural Reconstruction.

shown in physical space in the picture to the left in Figure 6.7. Once the cluster centers were determined using the Neural Gas routine, the data was locally decomposed using (2.4), and then local KL was performed on each domain. Each decomposition resulted in a single dimension capturing approximately 98 percent of the energy (see Table 6.4 for a sampling of the eigenvalues).

The inverses were trained using a 1-3-20 architecture. Once the charts were constructed, the original trajectory was tracked through the clusters, noting the cluster ordering and the direction the trajectory took through the one dimensional subspaces. We found that indeed, the trajectory was cyclic through the cluster centers (and spent an approximately equal amount of time in each region). The reconstructions (neural and linear) were obtained using the variant of local integration mentioned in Chapter 5. The neural reconstruction (the right picture of Figure 6.7) was able to maintain the smooth quality of the original, while the linear reconstruction (see the left picture of Figure 6.8) was choppy, the trajectory jumping from tangent space to tangent space. The linear reconstruction, shown as a trajectory in Fourier space (the last three modes), is also shown in Figure 6.8 as reference and the large jumps between tangent planes is very clear (recall the discussion in Chapter 1).

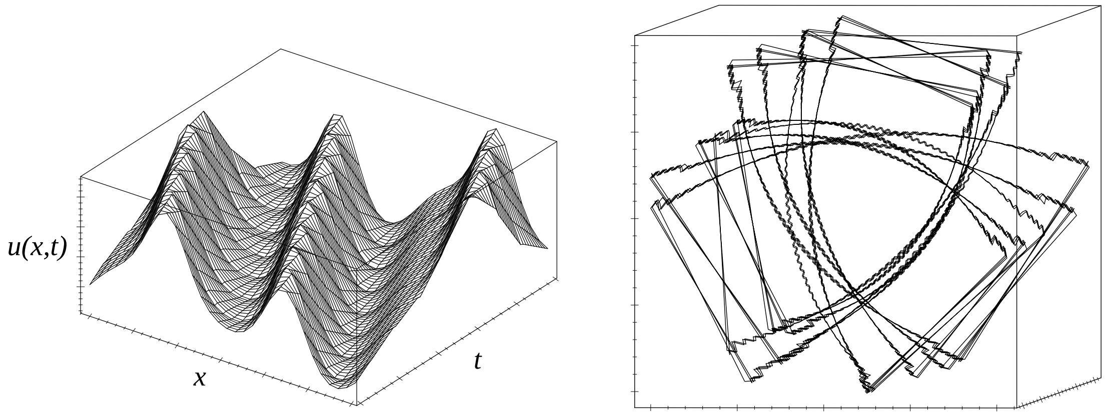

Figure 6.8: Linear reconstructions for $\alpha = 52.0$. Left: In Physical Space. Right: Last three Fourier modes

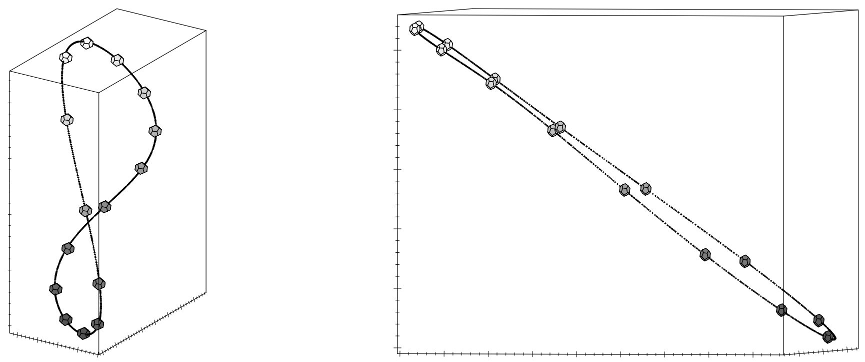

Figure 6.9: Fourier projections for $\alpha = 84.25$. Left: First three modes. Right: Last three modes. Also pictured are cluster centers.

Example: $\alpha = 84.25$

The attractor for this value of α is a cycle embedded in the Fourier space. As seen in the picture to the left in Figure 6.10, there is a large steady region of the flow, with more complicated, though periodic, motion on the two ends. It is this small mode motion that apparently makes it impossible to linearly compress this data (using global KL) to under 7 dimensions (see [47, 48], for example). Therefore, either a global, nonlinear technique or the nonlinear neural charts technique must be used. We have used a 15-cluster division of the cycle, and show in Figure 6.9 the result of three dimensional projections of the training data and cluster centers into the first three modes and last three modes of Fourier space. A local KL analysis

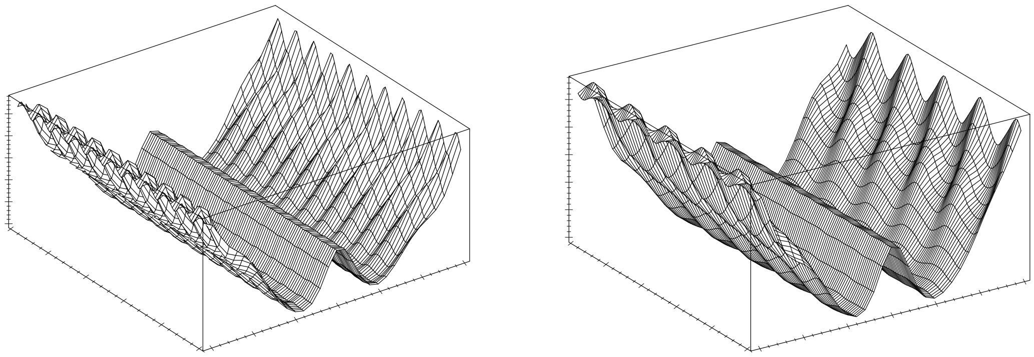

Figure 6.10: Physical space for $\alpha = 84.25$. Left: Original Data. Right: Neural Reconstruction (with half the time scale of the picture to the left).

was performed on each local region, obtaining one dimensional tangent spaces for each cluster. The divisions in the eigenvalues were very clear in this case (see Table 6.5 and compare to Table 6.4), probably due to the simpler nature of this attractor versus the previous one (i.e., compare the projections into Fourier space of the last three modes of these two attractors). The neural inverses were then trained, again using a 1-3-20 architecture. Analysis of the flow across the cluster

Table 6.5: Sample Eigenvalues: $\alpha = 84.25$

Cluster	Dim 1	Dim 2	Dim 3
0	0.998	0.001	0.0000
8	0.998	0.001	0.0000
13	0.995	0.004	0.0000

centers confirmed our expectation of a knot in Fourier space, and estimation of how long the flow spent in each cluster again provided approximately equal times.

The dynamical system was remodeled using both neural and linear reconstructions, again using the technique described in Chapter 5 for cyclic attractors, and the results are shown in Figure 6.10 (neural reconstruction to the right), and

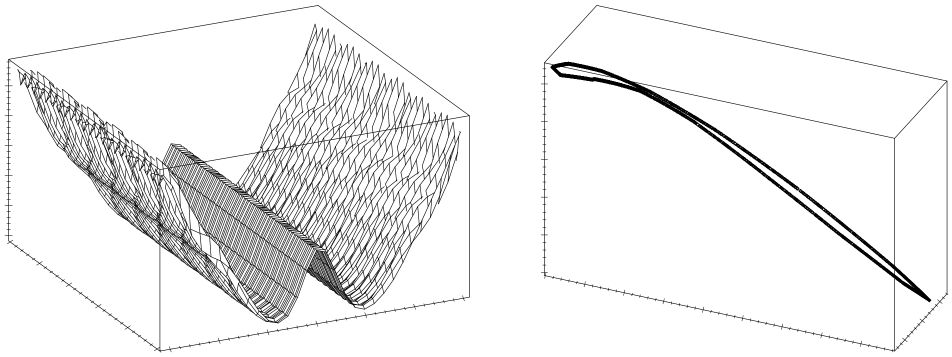

Figure 6.11: Linear reconstructions for $\alpha = 84.25$. Left: In physical space. Right: Last three Fourier modes.

Figure 6.11, where the linear reconstruction in physical space is shown to the left, and the Fourier projection (last three modes) is shown to the right. We again have recovered the smooth motion of the wave in the neural reconstruction, and while the linear reconstruction has maintained some of the global properties, the tangent plane jumps are still prominent.

Higher Dimensional Trajectories: $\alpha = 87$ **and** $\alpha = 68$

The following examples show the power of the neural charts algorithm and the ability of the new clustering algorithm, the Space-Time LBG, to seperate dynamical regimes that are close together in space. In this section, we are able to separate a "thin torus" into clusters that follow the dynamics — something that the standard clustering algorithms were unable to do. For the value $\alpha = 68$, we highlight the algorithm's ability to decompose a chaotic attracting set using both global and local models of the dynamics, and using both local linear and nonlinear decompositions. This culminates in a clear understanding of the dynamics and a confirmation of the results of the papers cited in this chapter.

Example: $\alpha = 87$

First, we do a global analysis of the data. The data represents a beating travelling wave, as seen in Figure 6.12 in physical coordinates, and in Figure 6.13

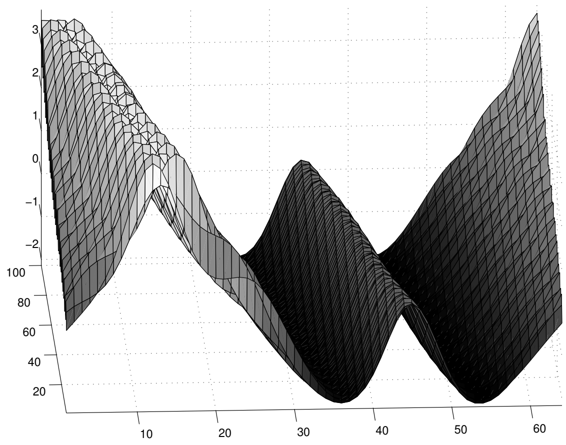

Figure 6.12: Original Data in Physical Space, $\alpha = 87$. This figure represents a beating, travelling wave.

Dim	Partial Sums				
1-5	0.3419	0.6719	0.7948	0.9142	0.9393
6-10	0.9644	0.9772	0.9899	0.9930	0.9960
11-15	0.9978	0.9996	0.9997	0.9998	0.9999

Table 6.6: Energy Distribution via normalized eigenvalues, $\alpha = 87.0$

in Fourier space. The images in Figure 6.13 represent the projections onto eigenvectors 1-3 and 4-6, respectively.

We performed a global KL expansion on the data, and the resulting normalized eigenvalues had a rather broad energy distribution, as seen in Figure 6.14. The partial sums of the normalized eigenvalues are given in Table 6.6. In order to retain 99% of the energy, we would need to project the data into no less than 9 dimensional space.

We now begin the clustering routine. Such a data set represents a difficult problem for the clustering routine, as the torus has such a thin radius. In fact, standard spatial routines formed a "necklace" through the torus, making it difficult, if not impossible, to formulate the local dimension (since local dimensions need to be based on the surface of the data). In Figure 6.15, we show the typical result of

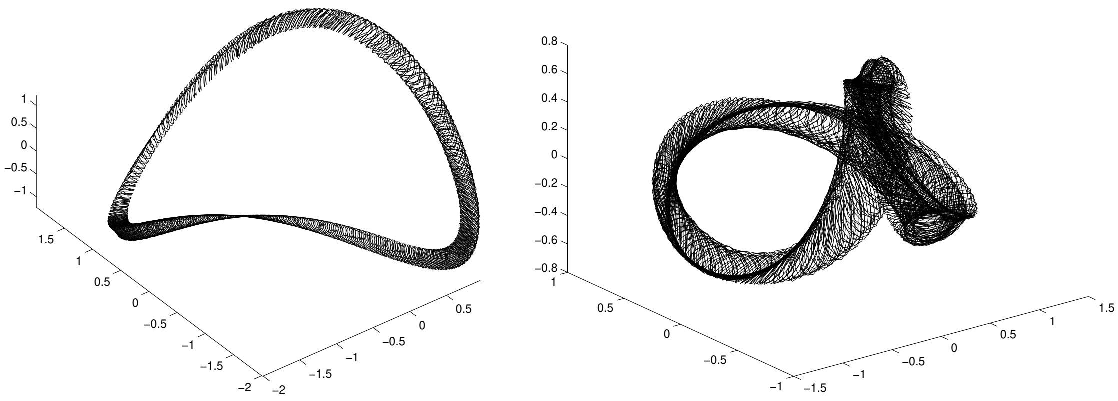

Figure 6.13: Original Data in Fourier Space, $\alpha = 87$. The first image is the projection to the first 3 eigenvectors, the second image is the projection to the next 3 eigenvectors.

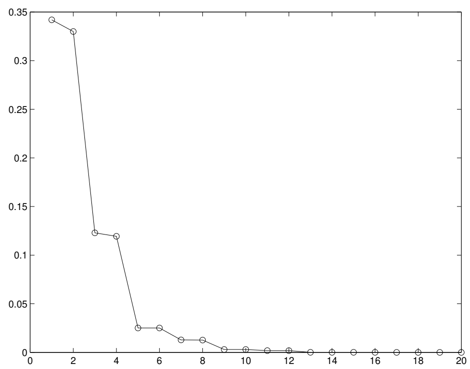

Figure 6.14: The normalized eigenvalues of the covariance matrix, $\alpha = 87$. Represents a rather broad global energy distribution.

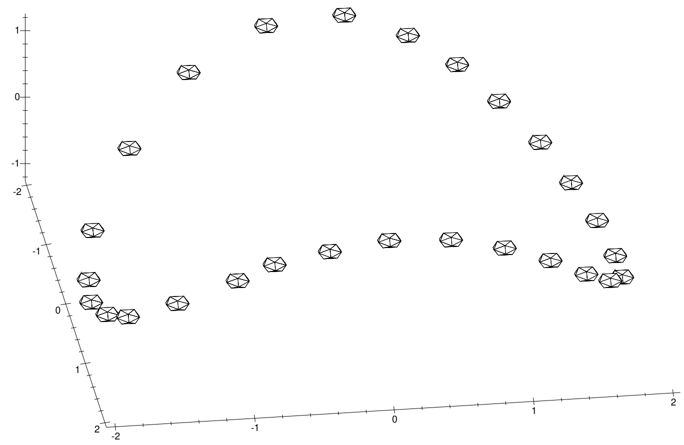

Figure 6.15: End result of Neural Gas clustering on the “thin torus”, $\alpha = 87$. The image shown is the projection into the first three dimensions of the data.

using a spatially based clustering algorithm on the data. In contrast, the Space-Time LBG algorithm gives a solution that is optimal for our task, and the result is given in Figure 6.16, with the data shown for reference (also see Section 2.5). The clustering algorithm produced 44 clusters at the terminal nodes of the tree.

The next stage is to determine the local dimensionality of the data. The Modified LSVD algorithm was used for this purpose, and the result of a sample cluster is given in Figure 6.17. The top image locates the cluster on the data, the bottom right image shows the local, whitened data in the first three dimensions, and the bottom left image shows the scaling of the singular values. First, the scalings do not start at the origin, since we needed to use a minimum of 21 points (since we are working in $\mathbb{R}^{20}$. See Sections 3.3 and 3.4). Secondly, there are clearly two singular values scaling linearly, thus the local dimension is two. The remaining singular values either scale quadratically, or scale at such a low value that they do not appear (having been removed in the whitening process). In other clusters, the dimensions did not scale as nicely. Figure 6.18 summarizes the results of the dimensionality estimation. Also recall from Chapter 5 that, with the RBF

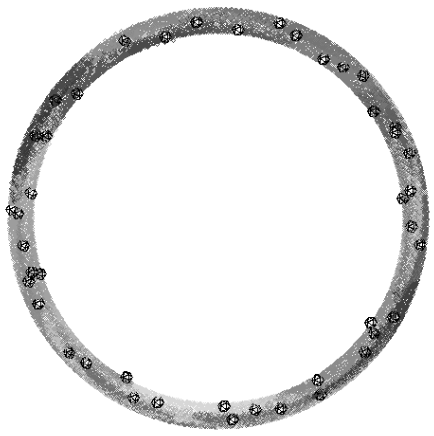

Figure 6.16: End result of Space-Time LBG clustering on the "thin torus", $\alpha = 87$. The image shown is the projection into the first three dimensions of the data. The clustering has followed the dynamics, staying on the surface.

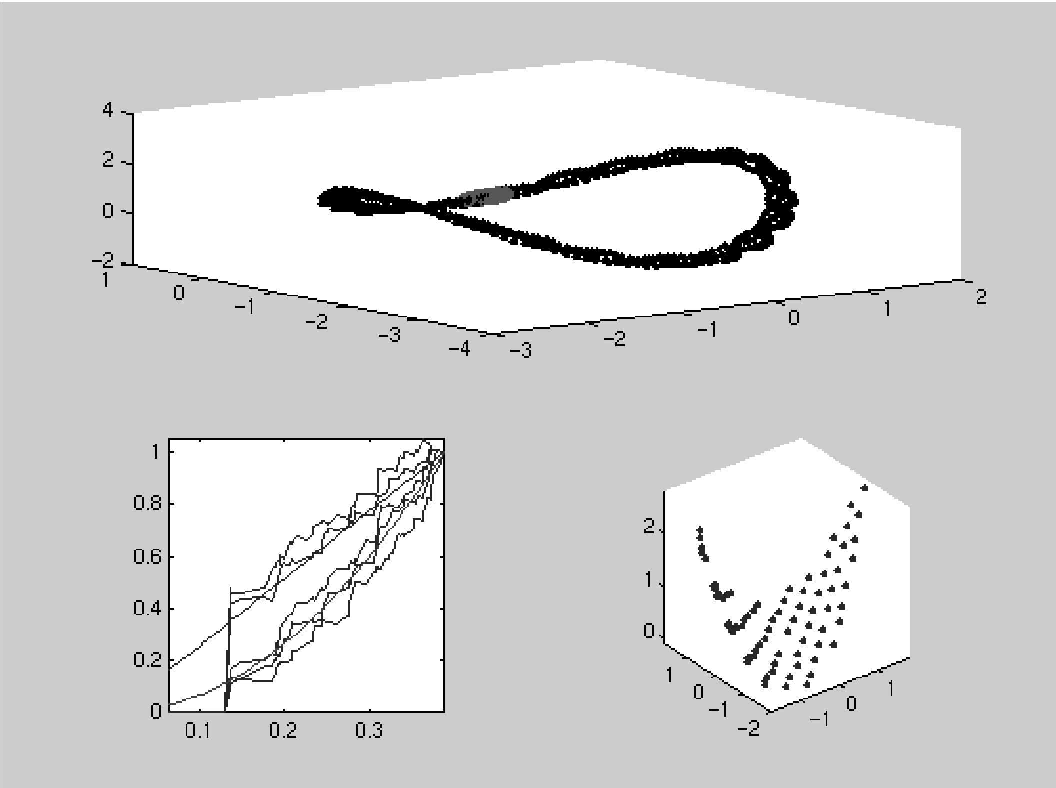

Figure 6.17: The Modified LSVD algorithm on a sample cluster from $\alpha = 87$. The top image locates the cluster, the bottom right image shows the local, whitened data (first 3 dimensions), and the bottom left image shows the scaling of the singular values. Note that there are two singular values scaling linearly.

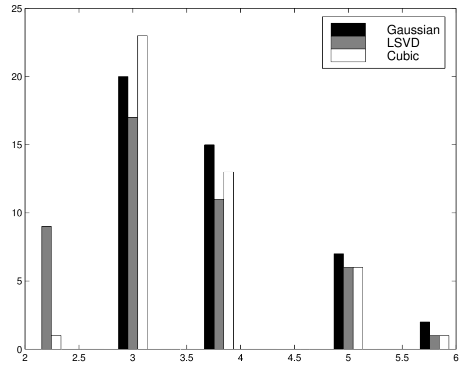

Figure 6.18: Histogram of the local dimensionality estimates, $\alpha = 87$. We compare the results of the Modified Local SVD (gray) with the estimates required by the RBF networks using Gaussian basis functions (black), and cubic basis functions (white).

networks, if we were unable to obtain a low enough error on the training set of data, then we increase the local dimensionality until the error is low enough. This discrepancy is also summarized in the table. We thus differentiate between the local dimension (which is what the Modified LSVD algorithm estimates), and the projection dimension (which the RBF uses to rebuild the data). These estimates are not contradictory, since one measure is strictly local (the Modified SVD), and the other is "global" with respect to the cluster. Using Gaussian basis functions for the RBF networks forced us to increase the dimensionality more often than when we used cubic transfer functions. The dimensionality required for the cubic RBF matched that for the LSVD in dimensions 5 and 6, while the Gaussians increased the dimensionality.

Of course, the crucial element to the neural charts algorithm is in the nonlinear reconstruction of the data. Training of the RBF networks utilized an expansion of the cluster as outlined in Chapter 5, with a removal of 20% of the data for validation so that the network could not memorize its input. We used the cubic transfer function, which generalized well on the manifold, as we saw in Figure

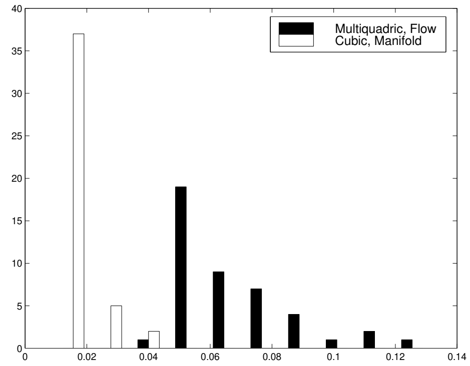

Figure 6.19: The relative sizes of the RBF Networks, using Equation (4.7). Here we compare the RBF sizes used for the manifold reconstruction (white, with cubic transfer functions) and the RBF sizes for the flow reconstruction (black, with multiquadric transfer functions).

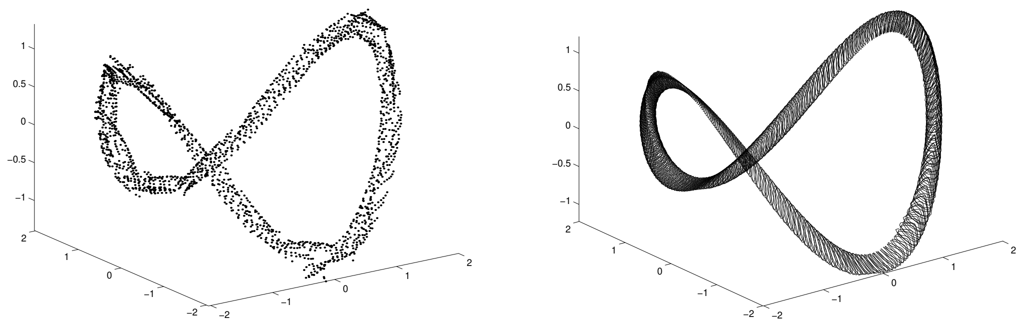

Figure 6.20: The linear reconstruction of $\alpha = 87$ is on the left. On the right is the neural reconstruction. Note the smooth transitions between clusters.

6.18. In Figure 6.19 we show the relative sizes using Equation (4.7). The RBFs for the manifold reconstruction were relatively quite small, indicatory of good generalization.

In Figure 6.20, we show the linear and neural reconstructions of the torus. As predicted, the neural reconstruction makes for smooth transitions between clusters.

We build the model of the flow using the neural approximation given at the end of Chapter 5. Multiquadrics were used as a comparison the cubics, with a

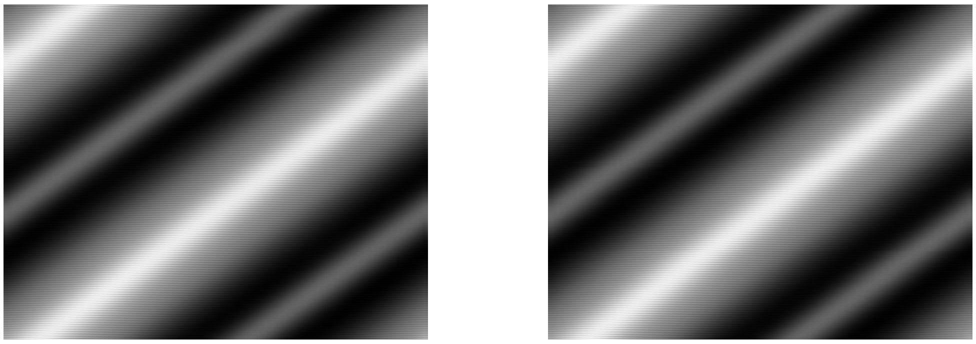

Figure 6.21: The neural reconstruction (right) of the flow versus the original (left). There is no noticeable difference.

training goal of attaining a MSE of 10^{-4}. As a side note, we first trained with a MSE of 10^{-3}, but the reconstruction of the dynamics resulted in a convergence to a fixed point similar to that at $\alpha = 72$. In a comparison of the relative sizes of the RBFs (Figure 6.19) needed to reconstruct the manifold versus reconstructing the flow shows that the RBFs needed to reconstruct the flow were more complex. This could also be the result of the different constructions of the domains and ranges. For the manifold, the neural network was learning the *error* between the linear reconstruction and the true data, while the flow RBF did not utilize a linear reconstruction for a starting value.

The images in Figure 6.21 compare the neural flow to the original flow, and there is no noticeable difference. The difference between the original flow and the original trajectory can be seen in Figure 6.22, where we plot the first 100 points of the trajectory in the 20th dimension. Although there are small errors, as we can see in Figure 6.23, where we plot the first 1000 points, the qualitative dynamics are the same.

We see that the RBF model of the flow, using the neural charts as a starting point, can quickly recover the dynamics of the system.

Example: $\alpha = 68.0$

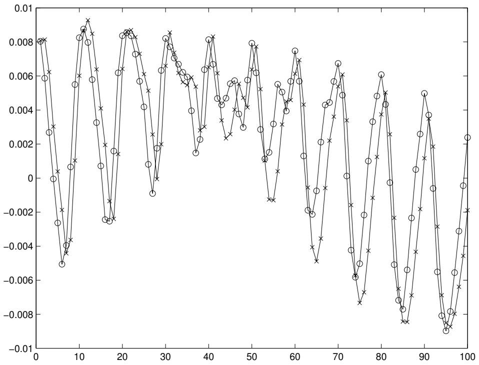

Figure 6.22: There are small errors seen in a close up of the reconstruction of the 20th dimension of $\alpha = 87$, first 100 points. The crosses highlight the original trajectory, and the circles the reconstructed values.

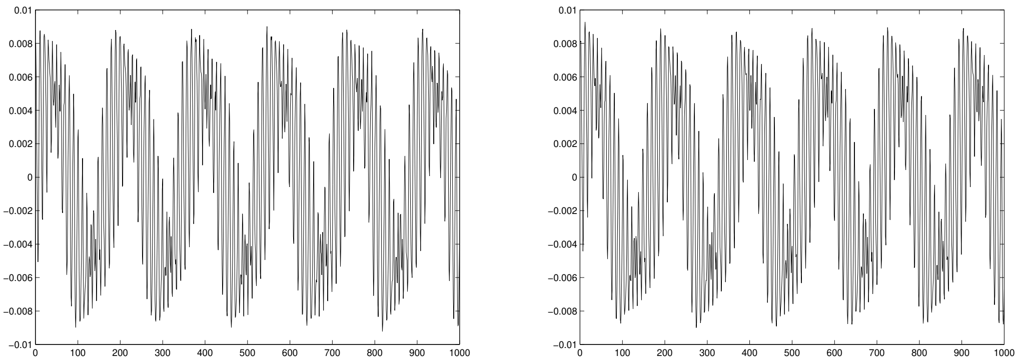

Figure 6.23: The image to the left is the neural reconstruction of the 20th dimension of the trajectory (1000 points), $\alpha = 87.0$. The image to the right is the original. The neural networks have captured the qualitative dynamics. Compare with Figure 6.22, which is a close up view of the first 100 points.

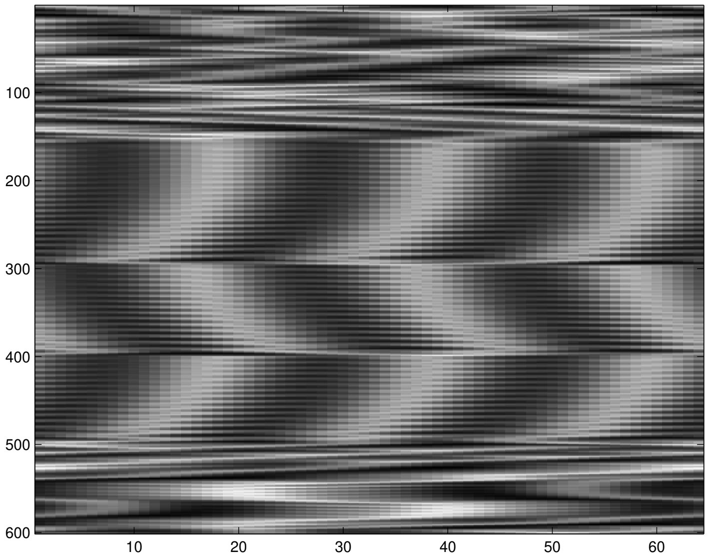

Figure 6.24: A snapshot of the dynamics in physical space, $\alpha = 68$. There is a basic motion of left travelling wave, right travelling wave and an excursion. The horizontal axis is the spatial direction, running from 0 to 2π in 64 increments. Time is along the vertical axis.

At $\alpha = 68$, we have complex oscillatory and chaotic behavior. A snaphsot of the dynamics appear in Figure 6.24 where we see that the motion globally consists generally of a left travelling wave, a right travelling wave, and bursting behavior.

These categories of motion can also be seen in Fourier space, where we project the data to the dominant 6 KL eigenvectors. The image to the left of Figure 6.25 shows the projection to the first 3 KL eigenvectors, and the image to the right shows the projection to the next 3.

In the global KL of the data, we see a large spread of energy and a pairing of the KL eigenvectors. Figure 6.26 shows the normalized eigenvalues for the full data set (on the left) and its derivatives (on the right). Interesting to note is the four dimensional KL subspace in the derivative data. If we were to perform a global KL compression to the data, we would require a minimum of 10 eigenvectors to maintain 99% of the energy. Table 6.7 shows the partial sums of the normalized

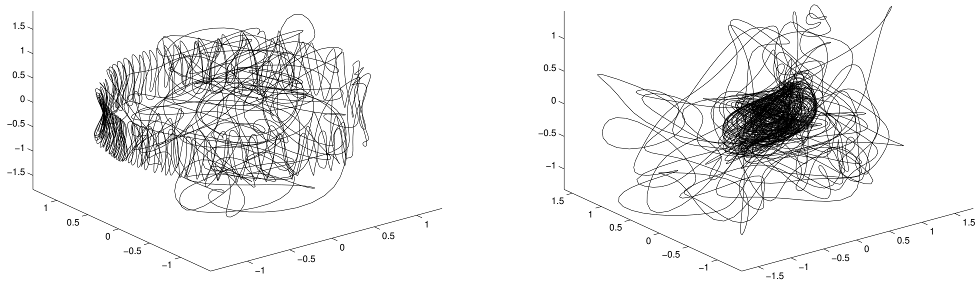

Figure 6.25: A snapshot of the dynamics in Fourier space, $\alpha = 68$. There is a basic motion of left travelling wave, right travelling wave and an excursion. The image to the left is a projection to the first 3 KL eigenvectors, and the image to the right is of the projection to eigenvectors 4-6.

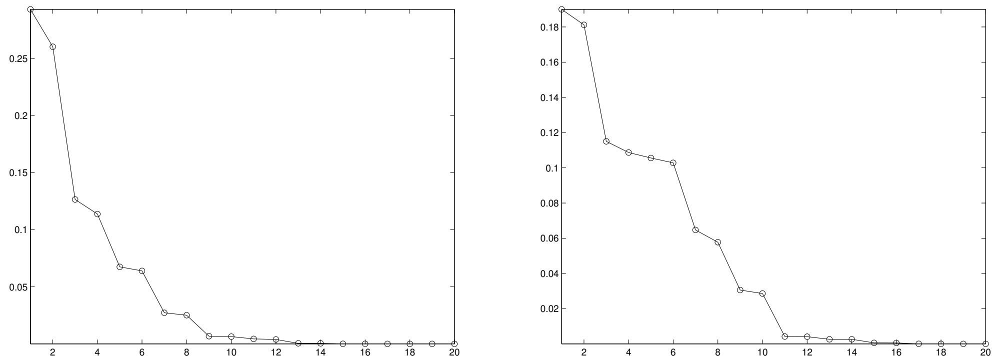

Figure 6.26: The normalized eigenvalues for the full data set, $\alpha = 68$, and its derivative data (on the right). Note the spread of the energy.

eigenvalues. From this, we can see that the primary three dimensional KL subspace (depicted in the left image of Figure 6.25) contains approximately 68% of the energy of the system.

We now construct a global model of the dynamics, using our previous observations that the motion can be decomposed generally into a left travelling wave (LW) set, a right travelling wave (RW) set, and an excursionary region. We found through several numerical tests that the LW-RW sets appear to be forming through

Dimension	Partial Sums				
1-5	0.2915	0.5513	0.6785	0.7923	0.8602
6-10	0.9247	0.9521	0.9774	0.9842	0.9906
11-15	0.9949	0.9988	0.9993	0.9998	0.9999

Table 6.7: Partial sums of the normalized eigenvalues, $\alpha = 68$. To capture 99% of the energy requires no less than 10 eigenvectors.

the third mode. The original data set was indexed by:

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_{20})^{\mathrm{T}}$$

so that the third mode is represented by x_5 and x_6. So we formulate:

$$x_5 + ix_6 = r\mathrm{e}^{i\theta} \quad \text{or} \quad \theta = \tan^{-1}\left(\frac{x_6}{x_5}\right)$$

so that $\dot{\theta}$ is given by:

$$\dot{\theta} = \frac{\dot{x_6}x_5 - \dot{x_5}x_6}{1 + \left(\frac{x_6}{x_5}\right)^2} \tag{6.4}$$

and

$$r = |x_5 + ix_6| \tag{6.5}$$

In what follows, we did not compute the denominator to $\dot{\theta}$, since we were interested in its sign and not necessarily in its size. Figure 6.27 shows the plot of $\dot{\theta}$ and r for a snapshot of the data. Upon investigation of these time series, we found that the LW and RW sets were formed via initial small, sign invariant oscillations in $\dot{\theta}$, followed by an increase in amplitude. There is a first sign change in $\dot{\theta}$, and this was found to be a trigger signalling the departure from the LW or RW set into the excursionary regime. A departure from the excursionary set into a LW or RW is initiated by a sustained, sign invariant and small oscillatory motion in $\dot{\theta}$, and a sustained, high value in r. These observations culminated in the following algorithm for the global model.

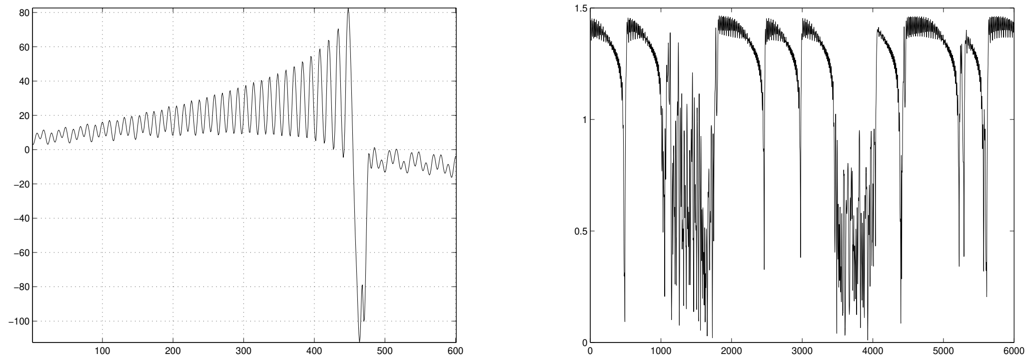

Figure 6.27: A snapshot of the time series for $\dot{\theta}$ and r (see Equations (6.4) and (6.5)). We use these to obtain a global breakdown of the trajectories into right wave, left wave and excursion.

Algorithm 6.1 *The Global Model for* $\alpha = 68.0$

1. Initialize 20 values of $\dot{\theta}$ and r.

2. Check to see if:

   (a) $\dot{\theta}$ changes sign.

   (b) $\dot{\theta}$ is large in size. (For this test, we used 20 as an ad-hoc criterion).

   (c) r is small in size. (For this test, 0.5 was an ad-hoc criterion).

3. If the three checks above are all false, then we are currently in a RW ($\dot{\theta}(20) < 0$) state, or a LW ($\dot{\theta}(20) > 0$) state. If this is the case, continue in the state until a first sign change occurs.

4. If the three checks above are not all false, then we remain in an excursionary state.

The resulting sets are shown in Figures 6.28, 6.29, and 6.30. They depict the LW, Excursionary and RW data sets in (original) dimensions 1, 2 and 3 (left images) and dimensions 4, 5, and 6 (right images). The algorithm succeeded in

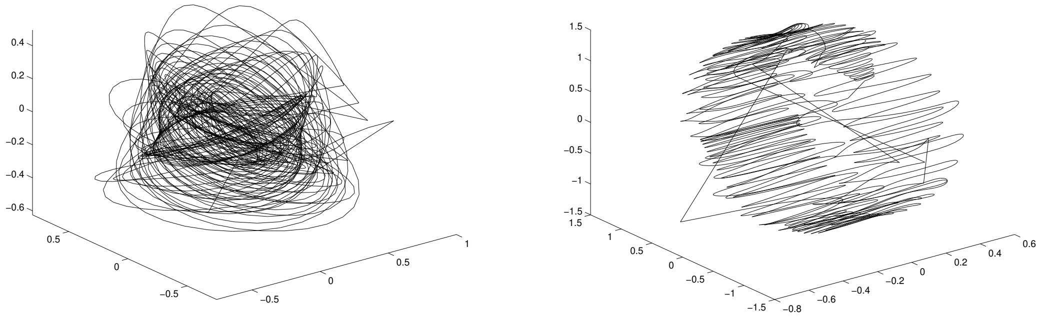

Figure 6.28: A snapshot of the dynamics in Fourier space, $\alpha = 68$. This is the left travelling wave in dimensions 1-3 and 4-6. The straight lines running through the right image are transients from the graphics package.

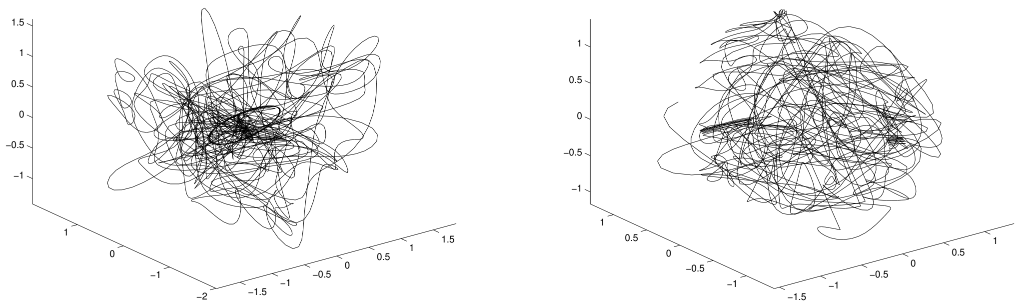

Figure 6.29: A snapshot of the dynamics in Fourier space, $\alpha = 68$. This is the excursionary regime, dimensions 1-3 and 4-6.

generally capturing the qualitative dynamics of each region, but also took in some parts of the trajectory that did not belong — but this is expected, since there is a smooth transition occurring between data sets. The decomposition resulted in approximately 47% of the data being classified as Excursionary, 32% was RW and 21% was LW.

Figure 6.31 shows the results of performing a KL expansion on the three data sets. As expected, the plot of the normalized eigenvalues for the LW and RW sets are almost identical (designated by crosses and circles, respectively), and have a sharp drop after the first two eigenvalues. In KL dimensions 3 and 4, there is

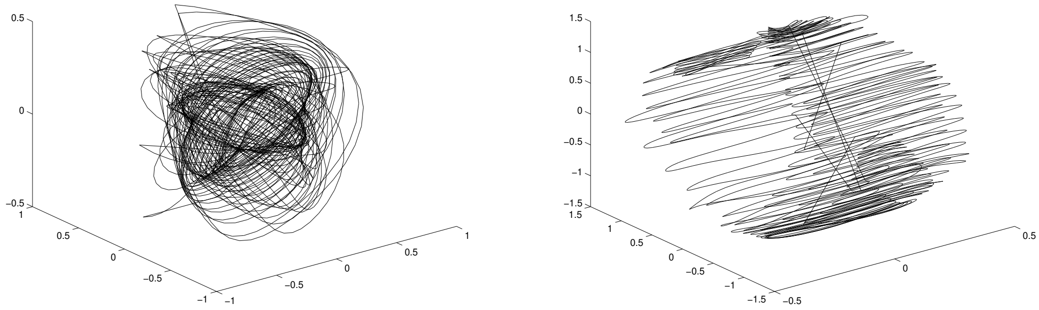

Figure 6.30: A snapshot of the dynamics in Fourier space, $\alpha = 68$. This is the right travelling wave in dimensions 1-3 and 4-6. The straight lines running through the right image are transients from the graphics package.

some transient energy remaining, and this will become important in the neural reconstructions to be performed later. In contrast, consider the KL spectrum of the excursionary data. There is a very strong 6 dimensional spread. We will revisit this data later after clustering.

The Space-Time LBG algorithm was applied to the LW and RW sets with the LBG parameter, α set to 0.5 (so that 50% of the metric was spatial and the other 50% was from the derivatives), and the algorithm produced 90 and 98 clusters in the terminal nodes respectively. The Space-Time LBG was attempted on the excursionary data (with non-zero contribution to the metric from the derivatives), but the derivatives of the clusters were such that the algorithm could not locate regions in phase space where the derivatives were approximately equal. This is highly suggestive of very complex, if not chaotic, dynamics in this region of phase space. The contribution to the metric from the derivatives was then set to 0, and the algorithm produced 134 clusters.

The clustering algorithm itself produced some interesting results on how the excursionary region (as well as the LW and RW sets) is organized. We consider each cluster to be a template of the local dynamics of the points within that cluster. Each cluster center therefore has a Fourier representation and a representation in

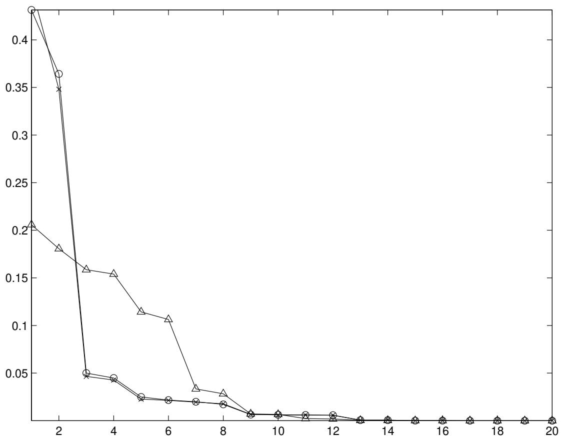

Figure 6.31: The normalized eigenvalues for the Left Wave (circles), Right Wave (crosses) and Excursionary (triangles) data sets. While the left and right wave sets have a large drop after the first two dimensions, the excursionary region has energy that is almost constant in the first four dimensions.

physical space. Since our attracting set is translationally invariant, we analyze the clusters using their Fourier power spectrum rather than the coordinates themselves. As expected, the Fourier spectrum of the clusters in the LW and RW set are rather uniform, with mode 3 being the most active. The image in Figure 6.32 depicts the active modes in the Fourier spectrum of the Left Wave set. The picture for the Right Wave set looks identical to this.

Contrast this image to the picture of the Fourier power spectrum of the excursionary clusters. They were ordered according to the size of the third mode, and the result is the image in Figure 6.33. We can now clearly see the mode interaction between modes 2 and 3. Futhermore, on the right side of the sorted clusters, we have a cluster whose image in physical space is shown in the right image in Figure 6.34. This is the remarkably similar to the strange fixed point at $\alpha = 72$, and on the left side, we have a cluster which is remarkably similar to the trimodal fixed point at $\alpha = 66$. The first 74 clusters (measuring left to right) contain about 47%

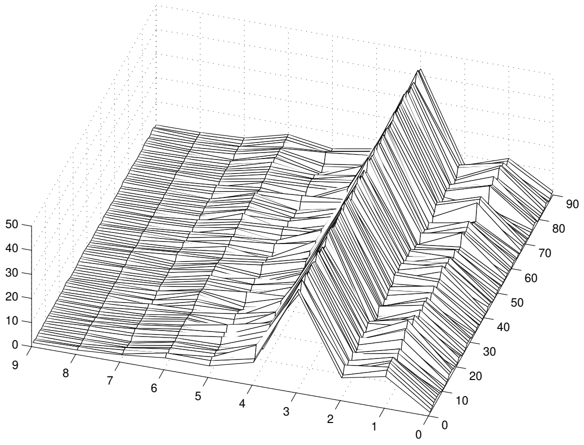

Figure 6.32: The Fourier power spectrum for the clusters in the Left Wave Set. They are uniformly most powerful in the third mode.

of the data, suggesting that approximately half the time, the trajectory is in the neighborhood of the strange fixed point, confirming the results in [47].

We also consider the dimensionality of the clusters in the excursionary region, ordered using the same index as in Figure 6.33. Dimensionality of each cluster was determined by using Fukunaga's criterion, with the energy cutoff ranging between 90% and 99.8%. The result is shown in Figure 6.35, where we averaged the dimension over 10 clusters each. Therefore, there are 13 sets of measures, indexed by the x-axis in the Figure. The lowest dimensionalities are measured in the neighborhood of the trimodal fixed point, while the remaining dimensionalities remain surprisingly consistent. Alternatively, we show a histogram of the dimensions versus the energy cutoff (the range is between 90% to 99.89% in increments of 0.001). Of note in this image is the concentration of dimensions in the low range.

We are now ready to perform a neural remodeling of the local manifold. As discussed in Chapter 5, we train the RBF network on the error between the linear reconstruction and the original data set. The error was then scaled to between 0 and 1, and the goal was to train the network to a MSE of 10^{-3} between the original data and the RBF output. We used cubic transfer functions, and kept back 20% of the data for validation. If the network was unable to attain the training goal, the

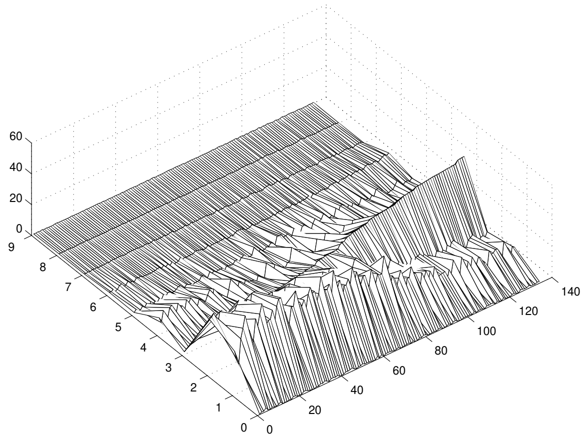

Figure 6.33: The Fourier power spectrum for the clusters in the excursionary set, ordered by the size of the third mode.

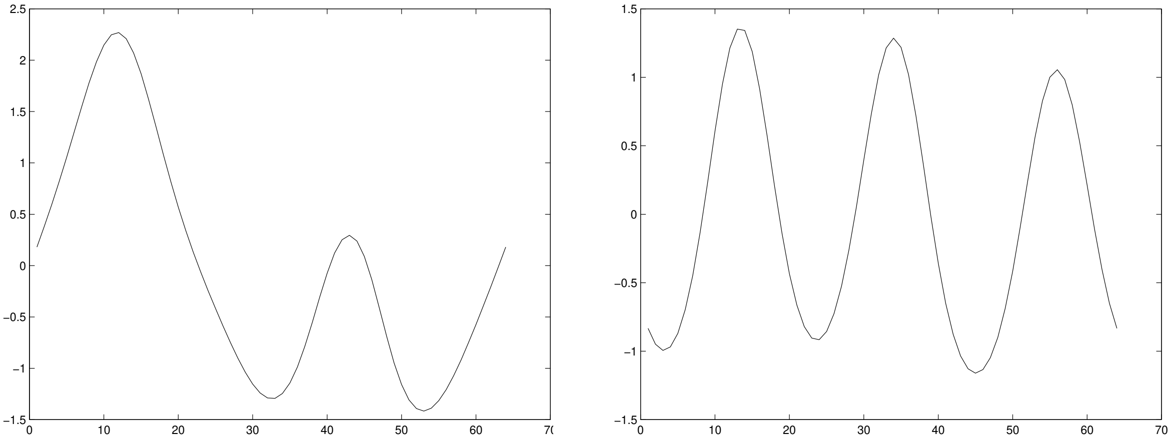

Figure 6.34: The physical representation of the clusters at either end of the power spectrum give in Figure 6.33. Compare with the strange fixed point, Figure 6.5!

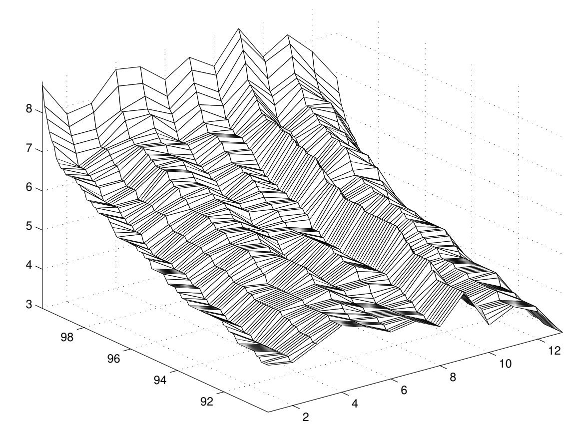

Figure 6.35: The dimensionality of the clusters in the excursionary region. The axis labelled 2-12 are the 13 means of the local clusters (10 clusters for the first 12, 14 clusters in the last). The energy cutoff is ranging from 90% to 99.8% in increments of 0.001. The vertical axis is the dimension.

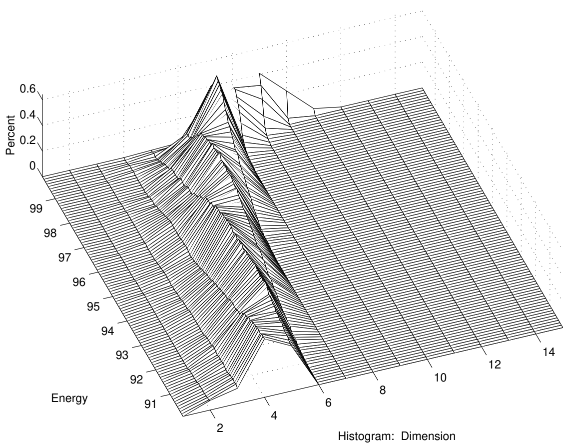

Figure 6.36: The dimensionality of the clusters in the excursionary region.

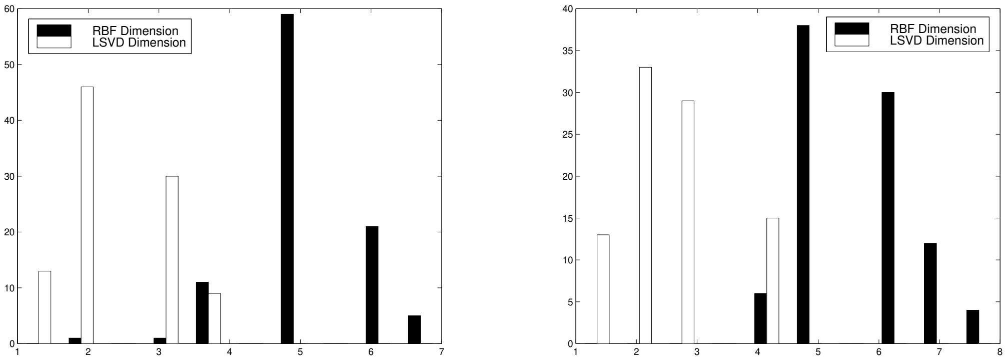

Figure 6.37: The differences in dimensionality estimates between the Modified Local SVD Algorithm (which estimates the dimension of the local tangent space), and the RBF dimension, which is the dimension necessary to attain the training goal.

local dimensionality was increased, and the network was allowed again to train. This continued until the RBF was successful. Again, we distinguish between the RBF dimension (the local dimension needed for the network to attain the training goal), and the Modified LSVD dimension, which is an estimate of the dimension of the local tangent space.

Figure 6.37 summarizes the difference between these dimensionalities. While the distribution is similar, the RBFs had a tendency to increase the local dimensionality. As discussed in the last example, this is not surprising since the task for the RBF is much different than the local SVD estimation. The subsequent neural remodeling of the data was excellent. To give a demonstration, we have included a close up of the data in the 18th dimension, the first 500 data points in Figure 6.38. The circles designate the original data, while the crosses mark the estimated values. Similar results were given in all of the clusters, for both RW and LW data sets.

RBF training also proceeded on the excursionary data, but it routinely increased the dimensionality to the point at which the linear reconstruction alone

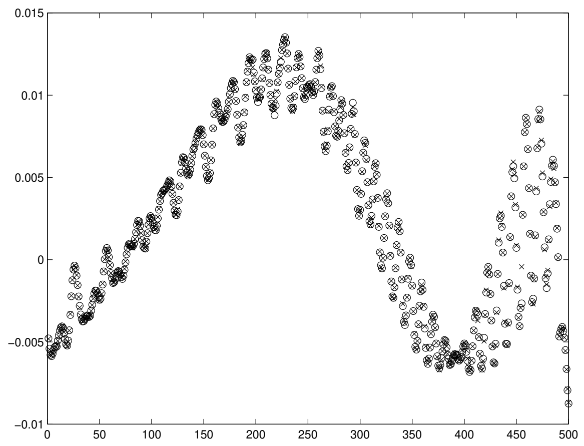

Figure 6.38: The neural reconstruction of the left wave, shown as a snapshot of the time series in dimension 18 of the original data. Even in this dimension, we have captured the dynamics. The original data are circles, the x's mark the reconstructed values.

attained the training goal (the dimensionality was uniformly either 11 or 12). Subsequently, we kept the excursionary clusters linear.

Lastly, for the LW and RW sets, we performed a neural modeling of the flow. The RBF domain and range for training are as described in Chapter 5, and we used the dimensionality found by the RBF networks for the manifold reconstruction. Multiquadric basis functions were used, and the measure of the complexity of the RBF network is computed as the ratio of the number of centers versus the number of data points used in training. The histogram of how complex these networks are is given in Figure 6.39. Of course, while the training error for the flow was a (global) MSE of 10^{-4}, the training error for the manifold RBF was a scaled 10^{-3}, so that we expect the manifold RBF networks to be slightly more complex. All networks maintained a relatively simple structure, taking no more than 40% of the training data as centers.

We used Algorithm 6.1 to model the global dynamics, with the following changes:

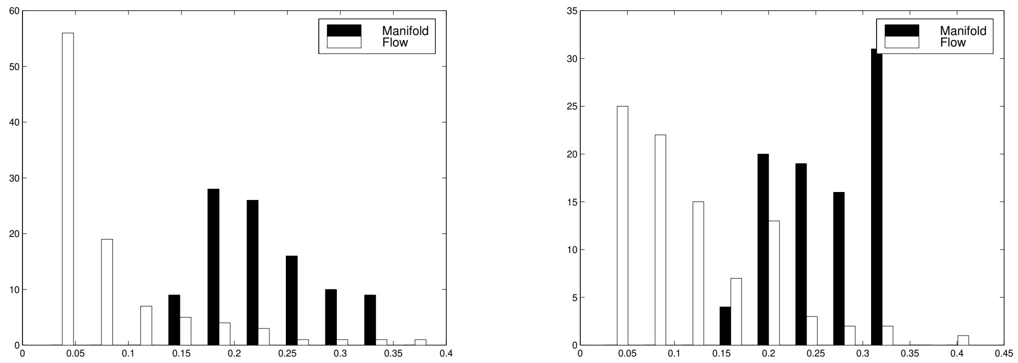

Figure 6.39: The complexity of the Manifold RBFs versus the Flow RBFs. The flow utilized multiquadric basis functions, and trained to an overall MSE of 10^{-4}, while the manifold RBF was cubic, and trained to a *scaled* error of 10^{-3}. All networks were relatively simple, using less than half the training data for centers.

1. If $\boldsymbol{v}$ is currently in the LW or RW set, we advance the solution by using the output of the RBF model.

2. If $\boldsymbol{v}$ is currently in the excursionary set, we advance the solution by using the Adams-Bashforth predictor corrector solver with local truncation error no greater than 10^{-6}. Since this results in much smaller step sizes than the RBF model, we allow the integrator to run for 250 time steps before continuing.

The results are shown in Figure 6.40, where the original solution is shown in the image to the left, and the neural reconstruction is shown on the right. The solutions are shown in the original 4-6 dimensions. Indeed, the interplay between the local and global models were able to accurately reconstruct the qualitative dynamics, as can be seen in the snapshot of the solution in physical space in Figure 6.41.

We can also compare the computational cost of using the RBF flow reconstruction versus integrating the Adams-Bashforth solver using the same time step.

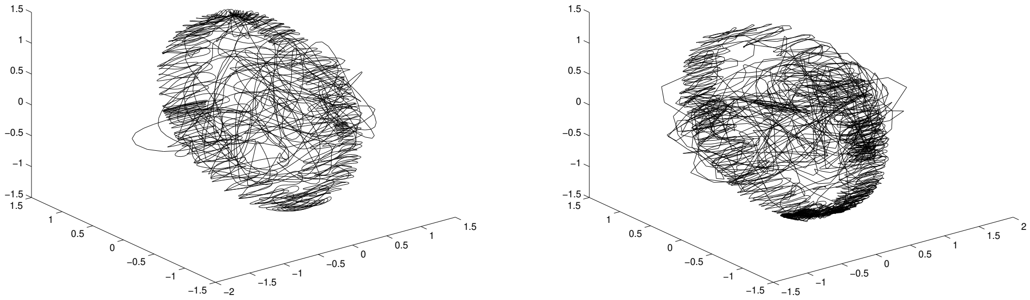

Figure 6.40: The original solution and the global-local model of the dynamics, given in the original 4-6 coordinate dimensions.

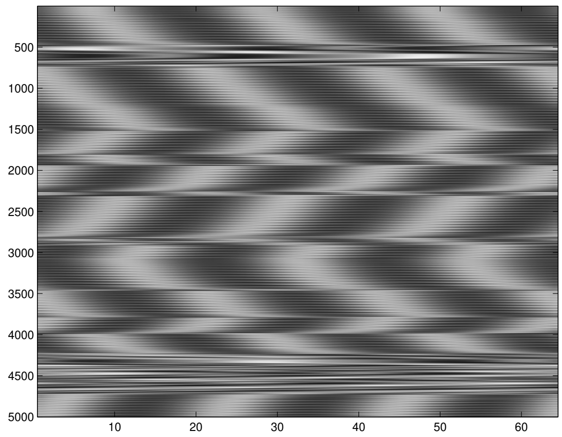

Figure 6.41: The output of the global-local model of the dynamics shown in physical space.

Algorithm	flops
RBF	7,543
A-B	551,043

Table 6.8: Floating Point comparison between the RBF reconstruction of one time step versus the Adams-Bashforth integration over approximately the same time step.

Table 6.8 shows the average number of floating point operations for advancing the solution 1 step in time. There is an overwhelming advantange to using the RBF reconstruction. The comparison was made where the integrator is repeated 250 times to move one time step, since the local truncation error was very small. In the case that the integration is not so delicate, the computational savings may not be so great.

6.4 Concluding Remarks

In this chapter, we have seen the power of the neural charts algorithm to locally decompose complex dynamical systems into small, dynamically homogeneous regimes where further local analysis can lead to insights into the dynamics. We have incorporated cutting edge techniques of data clustering, dimensionality estimation and neural approximation into the algorithm, which gives it the flexibility to incorporate prior knowledge of the structure into the model. In some cases, this prior knowledge is essential — therefore, the neural charts technique is not a black box method.

The failures of some of the algorithms can give us as much or more information as the successes. For example, we can see why the Space-Time LBG will ultimately fail in finding dynamically similar regions of phase space in the combination LW-RW data sets, and this failure led us to rethink how the model should be formulated. Futhermore, in the excursionary region, the derivative information was conflicting on the smallest of scales — suggesting very complex, if not chaotic motion. However, we saw that this did not stop our local analysis. The clustering information itself was valuable in understanding how the trajectories in the excursion were organizing themselves, and confirmed earlier results.

Examination and analysis of the local flow is thus dependent upon all of the pieces of the neural charts algorithm, and together they can tell us much about the dynamical system under study.

Chapter 7

DYNAMICAL REDUCTION MODELS: THE STATE OF THE ART

All of the techniques described below share a common characteristic. It is assumed that the dynamics described by

$$\dot{\mathbf{x}}(t) = F(\mathbf{x}(t)) \tag{7.1}$$

either lie on a low dimensional attracting set $\mathcal{A}$ or can be accurately projected onto $\mathcal{A}$. It is further assumed that the attracting set is embedded in a subset of $\mathbb{R}^N$ that has an artificially high dimensionality. We reduce the dimensionality of the dynamical system (7.1) by restricting the dynamics to $\mathcal{A}$. Given that this is the goal, the problem of reducing the system now becomes:

Find a low dimensional description of $\mathcal{A}$.

As was said previously, the attracting set is generally not itself a manifold. At this point, consider two methods of approximation to $\mathcal{A}$. We can either embed $\mathcal{A}$ into a global manifold, or we can embed $\mathcal{A}$ into local manifolds. Generally, the dimension of the global embedding will be much higher than the local embedding dimensions, as we can see later with the dimensionality estimates put onto the inertial manifold reduction. Every method we discuss is a method of approximation via the embedding into a manifold, so we discuss how the manifold will be represented next.

7.1 Manifold Representation

There are three general ways of expressing a manifold. The tactics one uses to perform the compression depends on which representation is chosen.

1. **A Global Embedding.** In this representation, the manifold is supposed to be k-dimensional, and currently it resides in $\mathbb{R}^n$. Then two differentiable functions are created,

$$G : \mathcal{A} \subset U^{\text{compact}} \subset \mathbb{R}^n \to V^{\text{compact}} \subset \mathbb{R}^k$$

and the approximate inverse to F:

$$H :\subset V^{\text{compact}} \subset \mathbb{R}^k \to U^{\text{compact}} \subset \mathbb{R}^n$$

Then the manifold is expressed globally as the $k-$ dimensional image of H, embedded in $\mathbb{R}^n$.

2. **The Graph of a Function.** In this case, consider the data set in $\mathbb{R}^n$ as $(x_1, x_2, \ldots, x_n)^{\mathrm{T}}$. We find a coordinate system and re-express the data as the graph of a function, so that

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = (y_1, y_2, \ldots, y_k, f_1(\boldsymbol{y}), f_2(\boldsymbol{y}), \ldots, f_{n-k}(\boldsymbol{y}))^{\mathrm{T}}$$

Thus, the data has been expressed as the graph of $\boldsymbol{f}$.

3. **A Set of Charts.** From the definition of a manifold, we get the concept of a set of charts. Since a manifold $\mathcal{M}$ is defined to be a set that is locally homeomorphic to $\mathbb{R}^k$, we must have local homeomorphisms. The set of charts defines these,

$$\pi_i : U_i^{\text{open}} \subset \mathbb{R}^n \to V_i^{\text{open}} \subset \mathbb{R}^k$$

for $i = 1 \ldots p$, and the preimages cover the manifold,

$$\mathcal{M} \subset \bigcup_{i=1}^{p} \pi_i^{-1}(V_i)$$

In the case that we are embedding an object $\mathcal{A}$ into $\mathcal{M}$, then this definition of the charts will force us to choose a value of k that will work for *all* neighborhoods. This implies that $\mathcal{M}$ is a global $k-$dimensional manifold, and in general, the value of k will be much higher than if we loosen the definition of the chart so that the dimension depends on the neighborhood;

$$\pi_i : U_i^{\text{open}} \subset \mathbb{R}^n \rightarrow V_i^{\text{open}} \subset \mathbb{R}^{k_i}$$

Note that now, the union of preimages will not define a global manifold. However, by definition, the preimages *will* be *local* manifolds, which will suffice for the construction of the functions π and π^{-1}, and will allow us to investigate the local dimensionality of the data set. Indeed, as we have seen in Chapter 6, the whole point of the analysis is to discover the dimensionality of the data as a local property. We do not need to be overly concerned as to the transition between the preimages of two charts, since the transition occurs on $\mathcal{A}$, and not between the local manifolds. Of course, as we mentioned previously in the neural charts routine, we expand the local neighborhoods so that our numerical approximations of the inverses are more accurate. Theoretically, we know the inverses exist, and the inverses, *when restricted to the attracting set*, have smooth transitions.

The state of the art techniques in dynamical reduction can generally be classified by how the attracting set will be represented. That this problem has a multitude of solutions is reflected in the diversity of current literature devoted to the topic. Here, we consider the leading techniques, and compare them to the neural charts algorithm.

7.2 Global Neural Reduction

Let us consider a novel technique first described in [49] that exploits the autoassociative neural network's ability to produce a global nonlinear embedding of the original dynamical system to one of smaller dimension. See Chapter 5 for a review of autoassociative (or bottleneck) neural networks.

Now, consider the first method of representing a manifold, where we express it as the image of a $k-$dimensional diffeomorphism. We are given data in $\mathbb{R}^n$ that is representative of the attracting set to the dynamical system, Equation (7.1).

A dimensionality k is preselected for the number of nodes in the bottleneck layer. This number can also be found by other methods (see [49] for more details). The autoassociative network then uses the data $\boldsymbol{x}^{(\mu)}$ for both input and output. Once trained, both functions G and H can be found by splitting the autoassociator apart.

The dynamical system in Equation (7.1) has the k-dimensional representation:

$$\dot{\mathbf{y}}_j(t) = \sum_{i=1}^{N} \frac{\partial G_j}{\partial x_i} F_i(\mathbf{x}(t))$$

where $j = 1, \ldots, k$. This is a complete semi-analytic low dimensional description that is topologically equivalent to the original system.

This is a global description, however, and doesn't yield local information which is what we will be after in the neural charts algorithm. Two other difficulties are associated with any technique that utilize the bottleneck architecture: setting the dimension of the bottleneck layer, and training the autoassociator. There are techniques to alleviate these difficulties, but can lead to very slow training. In comparison, the neural charts algorithm is designed to be much faster. The autoassociative neural network has also played a key role in other dimensionality reduction problems, for example see [21, 20, 62, 67].

The remaining techniques will all rely on a similar notation and setup, which is discussed next.

7.3 A General Framework

We now place the setting of the discussion in terms of a general framework used in [69].

Suppose the dynamical system is given by equation (7.1). For now, we will let $\mathbf{x}$ be either finite dimensional (in $\boldsymbol{R}^N$) or infinite dimensional (in the Hilbert space H). Let $\{\mathbf{w}_k\}$ be an (finite or infinite dimensional) orthonormal basis. Denote the projection onto the first M basis elements by:

$$P_M(\mathbf{x}) = \mathbf{x}_M = \sum_{k=1}^{M} (\mathbf{w}_k, \mathbf{x})\mathbf{w}_k \tag{7.2}$$

and

$$\mathbf{x}_M^C = (I - P_M)\mathbf{x}$$

so that $\mathbf{x} = \mathbf{x}_M + \mathbf{x}_M^C$. Then the original system of equations (7.1) is split by the projection defined above, yielding the new system:

$$\begin{aligned} \frac{d\mathbf{x}_M}{dt} &= P_M \frac{d\mathbf{x}}{dt} = P_M F(\mathbf{x}_M + \mathbf{x}_M^C) = Q_M(\mathbf{x}_M + \mathbf{x}_M^C) \\ \frac{d\mathbf{x}_M^C}{dt} &= (I - P_M)\frac{d\mathbf{x}}{dt} = (I - P_M)F(\mathbf{x}_M + \mathbf{x}_M^C) = R_M(\mathbf{x}_M + \mathbf{x}_M^C) \end{aligned} \tag{7.3}$$

Under the right circumstances, an approximation to the system is given by setting $\dot{\mathbf{x}}_M^C = 0$ and the dynamical system has been reduced to the first equation above. Below we consider these circumstances.

7.4 Neural Charts

Let the reduced dimensionality (referred to as M above) for the i^{th} cluster be denoted by M_i. The set $\{\mathbf{w}_i\}_{i=1}^{M_i}$ is then the basis for the tangent space. Thus the projection $P_{M_i} = \pi_i$, and $I - P_{M_i}$ is a projection into the orthogonal complement of the tangent space. We also have that $\mathbf{x} \approx \psi_i(\mathbf{v})$, where $\mathbf{v} = \mathbf{x}_{M_i}$ and $\mathbf{x}_{M_i} + \mathbf{x}_{M_i}^C \approx \psi_i(\mathbf{v})$ in the discussion above. Thus, equation (7.3) is of the form:

$$\frac{d\mathbf{x}_{M_i}}{dt} = Q_{M_i}(\mathbf{x}_{M_i} + \mathbf{x}_{M_i}^C) \tag{7.4}$$

$$\frac{d\mathbf{x}_{M_i}^C}{dt} = 0 \tag{7.5}$$

or,

$$\dot{\mathbf{v}} = Q_{M_i}(\psi_i(\mathbf{v}))$$

which is equivalent to equation (1.3). For comparison purposes, note that this method relies on the third method of expressing a manifold. This is a local, nonlinear technique.

7.5 Galerkin Projection

If the set $\{\mathbf{w}_k\}_{k=i}^M$ of basis vectors can be chosen so that $R_M \to 0$ asymptotically, and there exists aged initial data so that $\mathbf{x}_M^C(t)\mid_{t=0} = 0$ then $\mathbf{x}_M^C(t) \equiv 0$, and therefore equation (7.1) has been reduced to:

$$\dot{\mathbf{x}}_M = Q_M(\mathbf{x}_M) \tag{7.6}$$

The Galerkin method is the classical method. However, numerical experimentation has found that for many nonlinear systems of equations, this approximation is too crude without taking an excessive number of basis vectors. For example, we have discussed the strange fixed point for the KS Equation at $\alpha = 72$. Simulations at this value of α were performed in [47], computed both by using 10 complex modes, then with 9 complex modes in the Fourier expansion. While the dynamics in the 10 mode simulation were attracted to the strange mode, the 9 mode simulation approached the fixed point for only a short period of time (0.168 time units), then became unstable. Furthermore, as they reduced the number of dimensions used further, the time to instability decreased. Thus, to obtain a one dimensional attracting set, the Galerkin method required 10 complex modes (20 dimensions).

7.6 Local Adaptive Galerkin

This is a local linear decomposition based on first grouping the data into ϵ neighborhoods, performing a series of KL computations to determine the local dimensionality (see the discussion in Chapter 3, then obtaining the reduced description of the attracting set via a projection of the data to the local tangent spaces (approximated using the KL eigenvectors). The dynamical system is then remodeled using a local Galerkin procedure. If $\boldsymbol{b} = \boldsymbol{x}_{M_i}$, then the local equations are given by:

$$\dot{\boldsymbol{v}} = Q_{M_i}(\boldsymbol{b})$$

We have seen in the examples that the method using a local linear description may not accurately reproduce the qualitative nature of the flow without having to use a great number of clusters, and this number increases extensively with highly nonlinear flows. Other interesting aspects of this work include the conjecture (numerically substantiated) that the local tangent space dimension may be equal to the number of Lyapunov exponents λ (counted with multiplicity) which lie in the range $-\mu \leq \lambda \leq \mu$, where μ is the largest (positive) Lyapunov exponent.

Tracking with KL [47]

The Galerkin projection is used, but with basis vectors coming from the global KL. The analysis is local in time, so that the covariance matrix is now computed:

$$A(t) = \sum_{t=t_0}^{t_0-\delta t_0} \boldsymbol{x}(t)\boldsymbol{x}(t)^{\mathrm{T}}$$

This local method allowed them to separate bursting events and confirm what we also confirm in Chapter 6, that the excursions in $\alpha = 68$ are in the neighborhood of the strange fixed point from $\alpha = 72$.

Generally, this method allows one to track the trajectory and look for certain events that would otherwise be washed out (or averaged over) in the global KL. It also forms an optimal local basis that moves with the space, allowing one to look at the flow in terms of a moving frame of reference.

7.7 Nonlinear Galerkin (using Approximate Inertial Manifolds)

For the inertial manifold, let us first reformulate the problem slightly. Let

$$\dot{u} + Au + N(u) = 0$$

with u in the Hilbert space $\mathcal{H}$, and with A a linear self adjoint, positive operator with N a nonlinear lower order (norm is bounded above by A) operator. For example, the Kuramoto-Sivashinsky equation fits into this setting.

Let $\lambda_1 \leq \lambda_2 \leq \ldots$ be the eigenvalues of A with corresponding eigenvectors $\boldsymbol{w}_i$. We first compute a spectral gap [15] to find a dimension k so that the small eigenvalues stay small for all time.

As in the general framework, construct the projection P onto the first k eigenfunctions. Let $p = Pu$, $q = Qu$ be the projections of u onto the range of P and its complement, $Q = I - P$. Then

$$u = p + q$$

and the PDE will split:

$$\begin{aligned} \dot{p} + APp + PF(p+q) &= 0 \\ \dot{q} + AQq + QF(p+q) &= 0 \end{aligned} \tag{7.7}$$

The inertial manifold is then realized as the graph of a function $\Phi : P\mathcal{H} \to Q\mathcal{H}$, or

$$q = \Phi(p)$$

Then Equation (7.7) becomes uncoupled, and

$$\dot{p} + APp + PF(p + \Phi(p)) = 0$$

Formally, Φ has an infinite dimensional range, so let Φ_m be the m-dimensional representation of Φ. Then $\Phi_m \to \Phi$ as $m \to \infty$, and we obtain the finite system:

$$\dot{p} + APp + PF(p + \Phi_m(p)) = 0$$

There are many ways to approximate Φ_m, and this is an active area of research, both on theoretical grounds and in methodology and construction. For example, see [44, 55, 39, 65, 71, 15].

We outline one method below.

The approximation is made using Euler's method to obtain, for time step τ:

$$\hat{q} + \tau AQ\hat{q} + \tau QF(p + \hat{q}) = 0$$

so that

$$\hat{q} = -\tau(I + \tau AQ)^{-1}QF(p + \hat{q})$$

where we see that $\hat{q}$ is the fixed point of

$$\mathcal{F} : q \to -\tau(I + \tau AQ)^{-1}QF(p + q)$$

If τ is chosen to be comparable to λ_{k+1}^{-1}, then $\mathcal{F}$ is a contraction map. Furthermore, if q_0 is taken to be 0, then

$$\Phi_m(p) = -\tau(I + \tau AQ)^{-1}QF(p)$$

The general idea is thus to determine a function that slaves q to p. Note that this is also generally what the neural charts algorithm does (although our setup is a little different).

There is some concern as to when this method can be applied, for example see [42], although it has been proven that inertial manifolds do exist for the Kuramoto-Sivashinsky equation [15].

An Optimal Inertial Manifolds Basis

In [69], it is argued that the optimal basis for the inertial manifold is given by eigenvectors of the ensemble-averaged covariance matrix $\dot{X}^T\dot{X}$ rather than X^TX, yielding:

$$\dot{\mathbf{x}}_M = Q_M(\mathbf{x}_M + \mathbf{x}_M^C) \quad 0 = R_M(\mathbf{x}_M + \mathbf{x}_M^C) \tag{7.8}$$

It is suggested that, if $\mathbf{x}_M^C$ is small enough, then the relation above can be solved, although the solution is not given. Note that the neural charts concept does provide a local solution.

7.8 Wavelets

The properties that make wavelet basis analysis so attractive is that it yields a multiresolution (across both space and time parameters) basis. Although these basis elements are not optimal in the sense that the KL basis elements are optimal, it was shown that the loss of energy using the wavelet basis is not too great in comparison to the KL basis (summarized in [6]). However the multiresolution property makes this choice attractive if one is searching to localize certain properties of the flow in space-time. Implementation is similar to the Galerkin projection, once a suitable basis has been obtained. As was stated previously, this is a very active area of research, and we include only those papers dealing with use of wavelets in solving partial differential equations, for example see [29, 61, 7, 22]. As previously suggested, preliminary results suggest that in some cases, wavelet bases may not be optimal (see also [61]).

7.9 Conclusions

The current methods for dimensionality reduction of dynamical systems have been presented. In the details, we can see that the neural charts algorithm has a unique place, in that we have generalized the construction of the manifold in which the dynamics are present, and have constructed optimal local bases by which the manifold can be reconstructed relatively quickly using neural networks. In comparing algorithms, we see that the neural charts can be interpreted in a number of ways. In terms of inertial manifolds, our method is a local construction (via the global constructions currently in use). The local charts can also be interpreted as local bottleneck-style networks, where the front end of the network has been replaced by KL, and the back end replaced by the RBF. In terms of the optimal inertial manifolds basis, in constructing our clusters, we find local regions where the derivatives are (almost) constant in the Space-Time LBG. We also perform a type of heirarchical decomposition (as in wavelets) in constructing the tree of clusters, and our decomposition into the charts is locally optimal (in contrast to the wavelet decomposition).

Bibliography

[1] F. Amdjadi, P.J. Aston, and P. Plechac. Symmetry breaking Hopf bifurcations in equations with O(2) symmetry with application to the Kuramoto-Sivashinsky equation. *Journal of computational physics*, 131(1):181, 1997.

[2] N. Aubry, P. Holmes, J.L. Lumley, and E. Stone. Models for coherent structures in the wall layer. *J. Fluid Mech.*, 192:115, 1988.

[3] Pierre Baldi, Yves Chauvin, and Kurt Hornik. Back-propagation and unsupervised learning in neural networks. In Yves Chauvin and David E. Rumelhart, editors, *Backpropagation: Theory, Architectures and Applications*, Developments in Connections Theory, chapter 4, pages 99–135. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1995.

[4] Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Neural Networks*, 39(3):930–945, 1993.

[5] Andrew R. Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14:115–133, 1994.

[6] G. Berkooz, J. Elezgaray, and P. Holmes. Coherent structures in random media and wavelets. *Physica D*, 61, 1992.

[7] Gal Berkooz, Juan Elezgaray, Philip Holmes, John L. Lumley, and Andrew Poje. The proper orthogonal decomposition, wavelets and modal approaches to the dynamics of coherent structures. *Applied Scientific Research*, 53:321–338, 1994.

[8] Gal Berkooz, Philip Holmes, and John L. Lumley. Low-dimensional models of the wall region in a turbulent boundary layer: New results. *Physica D*, 58:402–406, 1992.

[9] Gal Berkooz, Philip Holmes, and John L. Lumley. The proper orthogonal decomposition in the analysis of turbulent flow. *Annu. Rev. Fluid Mech.*, 25:539–575, 1993.

[10] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, U.K., 1995.

[11] D. S. Broomhead, R. Jones, and G. P. King. Topological dimension and local coordinates from time series data. *J. Phys. A: Math. Gen*, 20:L563–L569, 1987.

[12] D.S. Broomhead, R. Indik, A.C. Newell, and D.A. Rand. Local adaptive Galerkin bases for large-dimensional dynamical systems. *Nonlinearity*, 4:159–197, 1991.

[13] D.S. Broomhead and David Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[14] S. Chen, C.F.N. Cowan, and P.M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[15] P. Constantin, C. Foias, B. Nicolaenko, and R. Temam. *Integral Manifolds and Inertial Manifolds for Dissipative Partial Differential Equations.* Springer-Verlag, 1989.

[16] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2:303–314, 1989.

[17] D. Armbruster, J. Guckenheimer, and P. Holmes. Kuramoto-Sivashinsky dynamics on the center-unstable manifold. *SIAM J. Appl. Math.*, 49:676, 1989.

[18] G. Dangelmayr and M. Kirby. Topology preserving maps for time-series prediction. *Unpublished Technical Report*, 1995.

[19] A. G. Darbyshire and D. S. Broomhead. Robust estimation of tangent maps and liapunov spectra. *Physica D.*, 89:287–305, 1996.

[20] David Demers. Dimensionality reduction for non-linear time series. In *Neural and Stochastic Methods in Image and Signal Processing*, volume SPIE 1766, 1992.

[21] David Demers and Garrison Cottrell. Nonlinear dimensionality reduction. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, number 5, pages 580–587, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[22] M. Farge, N. Kevlahan, V. Perrier, and E. Goirand. Wavelets and turbulence. *Proceedings of the IEEE*, 84(4):639–669, April 1996.

[23] Bernd Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, number 7, pages 625–632. MIT Press, 1995.

[24] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, Boston, MA, 1990.

[25] K. Fukunaga and D.R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, C-20(2):176–183, 1971.

[26] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–191, 1989.

[27] K. Funahashi. Approximate realization of identity mappings by three layer neural networks. *Electronics and Communications in Japan, Part 3*, 73(11):61–68, 1990.

[28] A. Gersho and R.M. Gray. *Vector Quantization and Signal Compression.* Kluwer, 1992.

[29] O. Goubet. Construction of approximate inertial manifolds using wavelets. *Siam J. Math. Anal.*, 23(6):1455–1481, November 1992.

[30] P. Grassberger and I. Procaccia. Measuring the strangeness of strange attractors. *Physica D*, 9:189, 1983.

[31] R. Gray. Vector quantization. *IEEE ASSP Magazine*, 1984.

[32] J. Hale and H. Kocak. *Dynamics and Bifurcations.* Springer, New York, 1991.

[33] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[34] Kurt Hornik. Some new results on neural network approximation. *Neural Networks*, 6:1069–1072, 1993.

[35] Kurt Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[36] Kurt Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3:551–560, 1989.

[37] D. Hundley, M. Kirby, and R. Miranda. Empirical dynamical systems reduction Part II: Neural charts. *To appear in Semi-analytic methods for the NSE, Proceedings of the CRM workshop, Montreal, Ed. K Coughlin.*

[38] D. Hundley, M. Kirby, and R. Miranda. Spherical nodes in neural networks with applications. In S.H. Dagli, B.R. Fernandez, J. Ghosh, and R.T. Soundar Kumara, editors, *Intelligent Engineering Through Artificial Neural Networks*, number 5. The American Society of Mechanical Engineers, 1995.

[39] F. Jauberteau, C. Rosier, and R. Temam. The nonlinear Galerkin method in computational fluid dynamics. *Applied Numerical Mathematics*, 6:361–370, 1989/90.

[40] J.M. Hyman, B. Nicolaenko, and S. Zaleski. Order and complexity in the Kuramoto-Sivashinsky model of weakly turbulent interfaces. *Physica D*, 23:265–292, 1986.

[41] J.M. Hyman and B. Nicolaenko. The Kuramoto-Sivashinsky equation: A bridge between PDE's and dynamical systems. Technical Report LA-UR 85-1556, Los Alamos National Laboratory, 1985.

[42] John G. Heywood and Rolf Rannacher. On the question of turbulence modeling by approximate inertial manifolds and the nonlinear galerkin method. *SIAM Journal on Numerical Analysis*, 30(6):1603, Dec 1993.

[43] I.T. Jolliffe. *Principal Component Analysis.* Springer, New York, 1986.

[44] M.S. Jolly, I.G. Kevrekidis, and E.S. Titi. Approximate inertial manifolds for the Kuramoto-Sivashinsky equation: Analysis and computations. *Physica D*, 44, 1990.

[45] Nanda Kambhatla and Todd K. Leen. Fast non-linear dimension reduction. In *Advances in Neural Information Processing Systems, No. 6*, pages 152–159, San Mateo, CA, 1994. Morgan Kaufmann Publishers.

[46] I.G. Kevrikidis, B. Nicolaenko, and C. Scovel. Back in the saddle again: a computer assisted study of the Kuramoto-Sivashinsky equation. *Siam J. Appl. Math.*, 50:760, 1990.

[47] M. Kirby and D. Armbruster. Reconstructing phase-space from PDE simulations. *Z. angew. Math. Phys.*, 43:999., 1992.

[48] M. Kirby and R. Miranda. Empirical dynamical systems reduction Part I: Global nonlinear transformations. *To appear in Semi-analytic methods for the NSE, Proceedings of the CRM workshop, Montreal, Ed. K Coughlin.*

[49] M. Kirby and R. Miranda. Nonlinear reduction of high-dimensional dynamical systems via neural networks. *Phys. Rev. Letters*, 72(12):1822, 1994.

[50] M. Kirby and R. Miranda. Circular nodes in neural networks. *Neural Computation*, 8(2):390–402, 1996.

[51] T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, Berlin, 1995.

[52] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantization design. *IEEE transactions on Communications*, 28(1):84–95, January 1980.

[53] E. Lorenz. Empirical orthogonal eigenfunctions and statistical weather prediction. Science Report No. 1, Statistical Forecasting Project 1, M.I.T., Cambridge, MA, 1956.

[54] J.L. Lumley. In *Atmospheric Turbulence and Radio Wave Propagation*, page 166, Nauka, Moscow, 1967. edited by A.M. Yaglom and V.I. Tatarski.

[55] M. Marion and R. Temam. Nonlinear Galerkin methods. *SIAM J. of Numer. Anal.*, 26:1139–1157, 1990.

[56] T. Martinetz and K. Schulten. A "neural-gas" network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 397–402. Elsevier Science Publishers, 1991.

[57] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[58] Thomas M. Martinetz, Stanislav G. Berkovich, and Kluas J. Schulten. "neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993.

[59] Charles A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constr. Approx.*, 2:11–22, 1986.

[60] John Moody and Christian Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[61] Mark Myers, Philip Holmes, Juan Elezgaray, and Gal Berkooz. Wavelet projections of the Kuramoto-Sivashinksy equation I. Heteroclinic cycles and modulated traveling waves for short systems. *Physica D*, 86:396–427, 1995.

[62] S. Nakagawa, Y.Ono, and Y. Hirata. Dimensionality reduction of dynamical patterns using a neural network. In B.H. Huang, S.Y. Kung, and C.A. Kamm, editors, *Neural Networks for Signal Processing I*, pages 256–265, 1991.

[63] E. Oja. Data compression, feature extraction, and autoassociation in feedforward neural networks. In T. Kohonen, K. Mäkisara., O. Simula, and J. Kangas, editors, *"Artificial Neural Networks"*, pages 737–745, NY, 1991. Elsevier Science Publishers.

[64] M.J.D. Powell. Radial basis functions for multivariable interpolation: A review. In J.C. Mason and M.G. Cox, editors, *Algorithms for Approximation*, pages 143–167. Oxford: Clarendon Press, 1987.

[65] James C. Robinson. Inertial manifolds for the Kuramoto-Sivshinsky equation. *Physics Letters A*, 184:190–193, 1994.

[66] J.D. Rodriguez and L. Sirovich. Low-dimensional dynamics for the complex Ginzburg-Landau equation. *Physica D*, 43:77, 1990.

[67] Eric Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intellignence*, 11(3):304–314, 1989.

[68] L. Sirovich. Turbulence and the dynamics of coherent structures, Parts I-III. *Quarterly of Applied Mathematics*, XLV(3):561, 1987.

[69] L. Sirovich, B.W. Knight, and J.D. Rodriguez. Optimal low-dimensional dynamical approximations. *Quarterly of Applied Mathematics*, XLVIII:535, 1990.

[70] S. Strogatz. *Nonlinear Dynamics and Chaos*. Addison Wesley, New York, 1994.

[71] Linda Sundbye. An introduction to infinite-dimensional dynamical systems and inertial manifolds. Technical Report PAM 103, University of Colorado, Boulder, Boulder, CO 80309, 1991.

Appendix A

LOCAL SVD ON THE SURFACE OF THE SPHERE

We use data on the surface of a sphere to illustrate the main ideas presented in Chapter 3. We will be computing the local dimensionality of a patch of data on the surface. Since the surface is assumed to have a uniform distribution of points, it will not matter where we take the center, so we will use the north pole, $(0, 0, 1)$. For the parametrization, let

$$\begin{aligned} x(\rho, \phi, \theta) &= \rho \sin(\phi) \cos(\theta) \\ y(\rho, \phi, \theta) &= \rho \sin(\phi) \sin(\theta) \\ z(\rho, \phi, \theta) &= \rho \cos(\phi) \end{aligned}$$

with

$$\rho = 1 \qquad 0 \leq \phi \leq \epsilon \qquad \text{and} \qquad 0 < \theta \leq 2\pi$$

Discrete Computations

For the discrete computations, we take the angle θ at random between 0 and 2π. In order to obtain a uniform distribution across the surface of the sphere, let k be a random number between -1 and 1, then take:

$$\phi = \arccos(k)$$

So that k is between $\cos(\epsilon) \leq k \leq \cos(0)$. Now, let $\{\theta_j\}_{j=1}^{N}$ be the discrete (uniform) sampling for θ, and let $\{k_i\}_{i=1}^{M}$ be the discrete (uniform) sampling that creates ϕ_i. We now form the covariance matrix:

$$K_{MN} = \frac{1}{M}\frac{1}{N}\sum_{i=1}^{M}\sum_{j=1}^{N}\mathbf{f}_{ij}(k_i, \theta_j) \tag{A.1}$$

where $k = cos(\phi)$, and

$$\mathbf{f}_{ij} = \begin{pmatrix} x_{ij}^2 & x_{ij}y_{ij} & x_{ij}z_{ij} \\ y_{ij}x_{ij} & y_{ij}^2 & y_{ij}z_{ij} \\ z_{ij}x_{ij} & z_{ij}y_{ij} & z_{ij}^2 \end{pmatrix} \tag{A.2}$$

where W_{ij} is the discretely sampled, center-subtracted coordinate function W. We will now reformulate this expression so that it's limiting expression is clearly the integral in polar coordinates. First, we construct Δk and $\Delta\theta$. Since these values are taken uniformly, we will feel free to re-express them as being taken at equal lengths, so that:

$$\Delta\theta = \frac{2\pi}{N}$$

and

$$\Delta k = \frac{\cos(0) - \cos(\epsilon)}{M} = \frac{1 - \cos(\epsilon)}{M}$$

We can now rewrite the sum in Equation (A.1) as:

$$K_{MN} = \frac{1}{2\pi(1 - \cos(\epsilon))}\sum_{i=1}^{M}\sum_{j=1}^{N}\mathbf{f}_{ij}(k_i, \theta_j)\Delta k\Delta\theta \tag{A.3}$$

Note for future reference that $2\pi(1 - \cos(\epsilon))$ is the area of the polar cap, when $\phi = \epsilon$. This term will appear often enough below that it will be convenient to rewrite. Let

$$A(\epsilon) = 2\pi(1 - \cos(\epsilon))$$

and since $k = \cos(\phi)$, then:

$$\Delta k = \sin(\phi)\Delta\phi$$

so that we rewrite Equation (A.3) as:

$$K_{MN} = \frac{1}{A(\epsilon)}\sum_{i=1}^{M}\sum_{j=1}^{N}\mathbf{f}_{ij}(\phi_i, \theta_j)\sin(\phi_i)\Delta\phi\Delta\theta \tag{A.4}$$

The Continuous KL

From above, it is clear that, as the mesh tends to 0, the sum represented in Equation (A.4) tends towards the Riemann Integral:

$$K_\infty = \frac{1}{A(\epsilon)} \int_0^{2\pi} \int_0^{\epsilon} \mathbf{f} \sin(\phi)\ d\phi d\theta$$

We compute the six integrals represented in the equation above to obtain the following values (to save space, let $dS = \sin(\phi)d\phi d\theta$):

$$\frac{1}{A(\epsilon)} \int_0^{2\pi} \int_0^{\epsilon} x^2\ dS = \frac{-\cos^2(\epsilon) - \cos(\epsilon) + 2}{6} \tag{A.5}$$

$$\frac{1}{A(\epsilon)} \int_0^{2\pi} \int_0^{\epsilon} y^2\ dS = \frac{-\cos^2(\epsilon) - \cos(\epsilon) + 2}{6} \tag{A.6}$$

$$\frac{1}{A(\epsilon)} \int_0^{2\pi} \int_0^{\epsilon} z^2\ dS = \frac{1}{3}(1 - \cos(\epsilon))^2 \tag{A.7}$$

$$\frac{1}{A(\epsilon)} \int_0^{2\pi} \int_0^{\epsilon} xy\ dS = 0$$

$$\frac{1}{A(\epsilon)} \int_0^{2\pi} \int_0^{\epsilon} xz\ dS = 0$$

$$\frac{1}{A(\epsilon)} \int_0^{2\pi} \int_0^{\epsilon} yz\ dS = 0$$

These integrals were parameterized with respect to the angular length. When we construct the local KL curves, we will be parameterizing with respect to radial length (i.e., $x \in X_t$ iff $\|x - (0,0,1)\| \leq t$, where we will let t expand from $\phi = 0$ to $\phi = \epsilon$). Therefore, we change coordinates from ϵ to t using the Law of Cosines:

$$\epsilon = \arccos\left(1 - \frac{t^2}{2}\right)$$

Substituting this value in the expressions in Equations (A.5, A.6, A.7), respectively yields:

$$\frac{-\cos^2(\epsilon) - \cos(\epsilon) + 2}{6} \Rightarrow \frac{1}{4}t^2 - \frac{1}{24}t^4$$

and

$$\frac{1}{3}(1-\cos(\epsilon))^2 \Rightarrow \frac{1}{12}t^4$$

which yields the (approximate) singular value matrix:

$$\begin{pmatrix} \frac{1}{2}t & 0 & 0 \\ 0 & \frac{1}{2}t & 0 \\ 0 & 0 & \frac{1}{\sqrt{12}}t^2 \end{pmatrix} \tag{A.8}$$

This states that the two primary singular values (corresponding to the eigenvectors whose span is the local tangent plane) will scale linearly with the radial distance. Since the curvature and distribution of points on the sphere is independent of where the center is, these singular value estimates should remain valid no matter where the center is chosen. We will see that these estimates are very good when the data is uniformly distributed, but not necessarily so when the data is not uniformly distributed.

Numerical Results

The figures below are the result of using the SVD algorithm. Figure (1) shows the three singular values as well as the curves $\frac{1}{2}t$ and $\frac{1}{\sqrt{12}}t^2$. Agreement between the discrete calculations and the continuous estimation is clearly very good, and we see immediately that the dimension of the tangent space to the sphere is two dimensional. In fact, this picture very closely resembles the local SVD taken with center at any point of the sphere, which is what we expect.

Discrete Computations: Non-Uniform Data

Critical to the analytic estimation was the uniformity of the data. Here, the data set here was constructed so that ϕ and θ are randomly selected points so that the distribution of points is heavier at the poles than the equator. Figures 2 and 3 depict the singular values for the center at $(0, 0)$ and $(\frac{\pi}{2}, \frac{\pi}{2})$, respectively. Also plotted are the line and parabola from the uniform estimates. What's striking is that now the slope of the linear scaling depends on where the center is taken. This dependence on distribution is not something we would like in an algorithm that is designed to automatically detect linear scalings.

The Jacobian of the Graph

In order to see the connections we make in the next appendix to the Jacobian of the graph, consider the computations below that show how the Jacobian and the local KL are related.

In order to write the sphere as the graph, the coordinate system used is:

$$\begin{pmatrix} \phi \\ \theta \\ \sin(\phi)\cos(\theta) \\ \sin(\phi)\sin(\theta) \\ \cos(\phi) \end{pmatrix}$$

so that the Jacobian matrix is given by:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ \cos(\phi)\cos(\theta) & -\sin(\phi)\sin(\theta) \\ \cos(\phi)\sin(\theta) & \sin(\phi)\sin(\theta) \\ -\sin(\phi) & 0 \end{pmatrix}$$

At $\phi = \frac{\pi}{4}, \theta = \frac{\pi}{4}$, we get the analytic covariance of the Jacobian, as $\epsilon \to 0$:

$$\begin{pmatrix} 1 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{\sqrt{2}} \\ 0 & 1 & -\frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & 0 & -\frac{1}{2\sqrt{2}} \\ \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & -\frac{1}{2\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & 0 & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2} \end{pmatrix}$$

Numerically, with $\epsilon = 0.01$ and $5,000$ data points, we obtain the matrix:

$$0.1671 \begin{pmatrix} 1.0000 & -0.0038 & 0.5018 & 0.4981 & -0.7071 \\ -0.0038 & 1.0137 & -0.5088 & 0.5049 & 0.0027 \\ 0.5018 & -0.5088 & 0.5053 & -0.0035 & -0.3548 \\ 0.4981 & 0.5049 & -0.0035 & 0.5015 & -0.3522 \\ -0.7071 & 0.0027 & -0.3548 & -0.3522 & 0.5001 \end{pmatrix}$$

So we see that, in taking the local KL, we are computing the span of the Jacobian matrix (in the right coordinate system).

Appendix B

KL EIGENVECTORS AND THE JACOBIAN MATRIX

The purpose of this section is to show that the process of taking local KL eigenvectors in an ϵ-ball about a point a and the process of determining the span of the tangent space at a are the same, as $\epsilon \to 0$. First, some notation.

Let the k-dimensional manifold $\mathcal{M} \subset \mathbb{R}^n$ be denoted by the graph of $f : \mathbb{R}^k \to \mathbb{R}^{n-k}$, so that $y \in \mathcal{M} \Rightarrow y = (x, f(x))$ for some $x \in R^k$. We will be approximating $\mathcal{M}$ at $(a, f(a))$ in a ϵ-ball, so denote

$$B_\epsilon(a) = \{(x, f(x)) \mid \|x - a\|_1 \leq \epsilon\}$$

Let $H : y \in \mathcal{M} \to R^{nxn}$ be the covariance matrix for the data in $\mathcal{M}$. Also, for future reference, note that:

$$\lim_{\epsilon \to 0} \frac{1}{(2\epsilon)^n} \int_{\|x\|_1 \leq \epsilon} \frac{xx^T}{\epsilon^2} dx = \frac{1}{3} I$$

The covariance of the data about $(a, f(a))$ is then given by:

$$\frac{1}{\text{Vol}(B_\epsilon(a))} \int_{B_\epsilon(a)} H(y) dS \tag{B.1}$$

with

$$dS = \sqrt{\det\left(I_{k\text{x}k} + Df^T(x) Df(x)\right)}\, dx \doteq \rho(x)\, dx$$

so, approximating ρ:

$$\rho(x) = \rho(a) + D\rho(a)(x - a) + O(\|x - a\|^2)$$

yields the first order approximation to Equation (B.1):

$$\frac{\rho(a)}{(2\epsilon)^k}\int_{\|x-a\|_1\leq\epsilon} H(x,f(x))\,dx$$

Now, if we divide by ϵ^2, we obtain:

$$\frac{\rho(a)}{(2\epsilon)^k}\frac{1}{\epsilon^2}\int_{\|x-a\|_1\leq\epsilon} H(x,f(x))\,dx \tag{B.2}$$

Now let $y_i = \frac{x_i-a_i}{2\epsilon}$. Then $dx = (2\epsilon)^k\,dy$, and Equation (B.2) becomes:

$$\frac{\rho(a)}{(2\epsilon)^k}\frac{1}{\epsilon^2}\int_{\|y\|_1\leq y} H(2\epsilon y+a, f(2\epsilon y+a))\,(2\epsilon)^k dy = \tag{B.3}$$

$$\frac{1}{\epsilon^2}\int_{\|y\|_1\leq y} H(2\epsilon y+a, f(2\epsilon y+a))\,dy \tag{B.4}$$

We now consider

$$\frac{H(2\epsilon y+a, f(2\epsilon y+a))}{\epsilon^2} = \tag{B.5}$$

$$\begin{pmatrix} 2\epsilon y \\ f(2\epsilon y+a)-f(a) \end{pmatrix}\begin{pmatrix} 2\epsilon y^T & (f(2\epsilon y+a)-f(a))^T \end{pmatrix} \tag{B.6}$$

We make another first order approximation:

$$2\epsilon Df(a)y \approx f(2\epsilon y+a)-f(a)$$

so that Equation (B.6) becomes:

$$\begin{pmatrix} 4yy^T & 4yy^TDf^T(a) \\ 4Df(a)yy^T & 4Df(a)yy^TDf^T(a) \end{pmatrix}$$

The integral in Equation (B.4) is therefore approximated by, to first order:

$$C\begin{pmatrix} I & Df^T(a) \\ Df(a) & Df(a)Df^T(a) \end{pmatrix}$$

where C is constant. Note that the span of this matrix is

$$\begin{pmatrix} I \\ Df(a) \end{pmatrix}$$

so that the local KL is approximating the span of the Jacobian matrix (in the KL coordinate system).

Appendix C

MATLAB AND RADIAL BASIS FUNCTIONS

The purpose of this chapter is to demonstrate the Matlab[1] implementation of the radial basis function solver. This will be worthwhile in two respects. We first see how the orthogonal least squares algorithm is extended to the multidimensional case, and also will learn some of Matlab's vectorization techniques.

If we have p centers, m points of dimension n as input data, and m points of dimension k for output data, then the training the RBF network is equivalent to solving the following equation for $\boldsymbol{A}$:

$$\Phi \boldsymbol{A} = \boldsymbol{Y} \tag{C.1}$$

where $\Phi_{i,j}$ is the ϕ acting on the i^{th} data point using the j^{th} center:

$$\Phi_{i,j} = \phi(\|\boldsymbol{x}^i - \boldsymbol{c}_j\|)$$

so that Φ has dimensions m by p.

The matrix values in A are the unknown coefficients, specifically, $A_{i,j}$ is the coefficient for the i^{th} center, j^{th} dimension, α_i^j, so that A has dimensions p by k.

Finally, $\boldsymbol{Y}$ is the target matrix, with dimensions m by k.

---

[1]Matlab is the property of Mathworks, Inc. The algorithms presented here are written by Mark Beale and are copyrighted by Mathworks.

The Solution to the RBF Equation

Given the transfer function ϕ and a list of centers, $\boldsymbol{c}_i$, then the solution to Equation (C.1) is:

$$\boldsymbol{A} = \Phi^{+}\boldsymbol{Y}$$

where Φ^{+} is the Moore-Penrose pseudoinverse. There are several methods available to solve for the pseudoinverse of a matrix. Two methods implementable in Matlab are given below:

1. The "pinv" command:

   $$\Phi^{+} = \mathrm{pinv}(\Phi)$$

2. The Singular Value Decomposition, Case I: Φ is full rank.

   Let Φ be m by p. Then the reduced singular value decomposition of Φ is given by:

   $$\Phi = USV^{\mathrm{T}}$$

   which in Matlab is produced by the command:

   $$[U, S, V] = \mathrm{svd}(\Phi, 0)$$

   where U is an m by p unitary matrix ($U^{\mathrm{T}}U = I$), V is an p by p unitary matrix, and S is an p by p diagonal matrix:

   $$S = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ 0 & \vdots & \dots & \vdots \\ 0 & \dots & & \sigma_p \end{pmatrix} \qquad S^{-1} = \begin{pmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ 0 & \vdots & \dots & \vdots \\ 0 & \dots & & \frac{1}{\sigma_p} \end{pmatrix}$$

   The pseudoinverse of Φ is then:

   $$\Phi^{+} = VS^{-1}U^{\mathrm{T}}$$

   In this case, either "pinv" or this SVD routine gives adequate numerical values. The special case is when Φ is NOT full rank.

3. Singular Value Decomposition, Case II: Φ is not full rank.

   In this case, S only has k non-zero singular values (or in practice, only k values above a "noise floor"). We note that the Matlab SVD routine returns singular values in descending order, so that the largest singular value is σ_1. Therefore, S has a diagonal k by k submatrix, and the rest of the matrix should be set to 0.

   If Φ is not full rank, then the "pinv" command has a default to choose the rank. This is why the singular value decomposition is sometimes preferable; you directly choose the values for which $\sigma_i = 0$. Once the noise floor cutoff has been determined, set k equal to the number of singular values above the cutoff. Then the psuedo inverse is given by (I'm using Matlab notation here):

$$\Phi^{+} = V(:, 1:k) * S^{-1}(1:k, 1:k) * U(:, 1:k)'$$

So the idea is that once the transfer function is chosen and the appropriate centers are found, then training the Radial Basis Function Network is just computing the pseudoinverse Φ.

Current Matlab Implementation

The following discusses the suite of commands that Matlab currently uses to solve the interpolation problem given the centers. Note that this is *not* the solverb routine, which finds the centers.

First, the Matlab implementation is transposed from our original setup. That is,

$$\Phi \tilde{\boldsymbol{A}} = \tilde{\boldsymbol{Y}} \Rightarrow \mathtt{AP} = \mathtt{Y}$$

Furthermore, the interpolation matrix is appended with a column (for Φ, or a row for P):

$$\Phi \Rightarrow \left[\Phi \middle| \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right]$$

In Matlab, let M by N be the size of P:

```
P = [P;ones(1,N)]
```

and the matrix A is split into two pieces:

```
A=[w2 b2]
```

The following commands mimic the approximation to Y:

```
a2 = ( Y / P) * P;
```

In solverb, we see the following suite of commands:

- Construct P using radbas. The constant b is for the Gaussian:

  ```
  P=radbas(dist(p',p)*b)
  ```

- Find the matrix A:

  ```
  [w2,b2]=solvelin(P,Y);
  ```

- Construct the approximation to Y:

  ```
  a2=purelin(w2*P,b2);
  ```

We now discuss the issues associated with Radial Basis Function interpolation: Finding a suitable transfer function and finding suitable centers.

Choosing the Centers

In a lot of research work, centers are chosen randomly from the data. It is also possible to perform clustering on the data, then choose centers from the clusters (a.k.a. codebook vectors). This method is not optimized for this purpose, however, and can be computationally expensive.

For many applications, randomly choosing the centers may be sufficient. In what follows, we apply a method known as Orthogonal Least Squares to choose the centers from the data. This algorithm is implemented in the Matlab routine "solverb", which is also discussed below.

The Matlab Implementation

We now discuss the Matlab implementation of this algorithm. Keep in mind that the default number of centers in "solverb" is the number of data points. This can cause some problems since the first iterate of the interpolation matrix has dimensions *Number of points* by *Number of points*!

Again, denote the interpolation matrix by its columns:

$$\Phi = [\Phi_1, \ldots, \Phi_p], \quad \Phi_i \in \mathbb{R}^n$$

which in Matlab will be denoted by P, and the output matrix, $\boldsymbol{Y}$ in terms of its rows:

$$\boldsymbol{Y}^{\mathrm{T}} = \begin{pmatrix} y_1^{\mathrm{T}} \\ \vdots \\ y_n^{\mathrm{T}} \end{pmatrix} = [\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k] = \boldsymbol{d}$$

So that $y_i \in \mathbb{R}^k$, and $\boldsymbol{d}_i \in \mathbb{R}^n$.

We are now ready to construct the algorithm. Key is the computation of error:

$$\frac{(\Phi_k^{\mathrm{T}} \boldsymbol{d}_j)^2}{\boldsymbol{d}_j^{\mathrm{T}} \boldsymbol{d}_j \Phi_k^{\mathrm{T}} \Phi_k} \tag{C.2}$$

which again is called the error associated with using the k^{th} column of Φ for the j^{th} data point.

Remark *First, we construct the matrix of elements for the denominator:*

Let

$$\mathtt{PP} = \begin{pmatrix} \Phi_1^{\mathrm{T}}\Phi_1 \\ \vdots \\ \Phi_m^{\mathrm{T}}\Phi_m \end{pmatrix} = \mathtt{sum}(\mathtt{P}.*\mathtt{P}')'$$

and

$$\mathtt{dd} = \begin{pmatrix} \boldsymbol{d}_1^{\mathrm{T}}\boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{d}_k \end{pmatrix} = \mathtt{sum}(\mathtt{d}.*\mathtt{d})'$$

Now the matrix of elements needed for the denominators of the error calculation in Equation (C.2) is:

$$\mathtt{dd}*\mathtt{PP}' = \begin{pmatrix} \boldsymbol{d}_1^{\mathrm{T}}\boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{d}_k \end{pmatrix} \left(\Phi_1^{\mathrm{T}}\Phi_1 \ldots \Phi_m^{\mathrm{T}}\Phi_m\right)$$

The $(j,k)^{\text{th}}$ element of this matrix is given by:

$$\boldsymbol{d}_j^{\mathrm{T}}\boldsymbol{d}_j\Phi_k^{\mathrm{T}}\Phi_k$$

Remark *We now construct a matrix that has the numerator elements from Equation (C.2).*

First, note that:

$$\begin{pmatrix} \Phi_1^{\mathrm{T}} \\ \vdots \\ \Phi_m^{\mathrm{T}} \end{pmatrix} (\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k) = \Phi^{\mathrm{T}}\boldsymbol{d} = \mathtt{P}'*\mathtt{d}$$

so that, in Matlab, the $(j,k)^{\text{th}}$ component of

```
((P'*d)'.^2)
```

is the numerator we needed:

$$\left(\Phi_k^{\mathrm{T}} d_j\right)^2$$

In conclusion, we construct the error value in Equation (C.2) by computing:

```
( (P'*d)'.^2 ) ./ (dd * PP')
```

Other Matlab ”Tricks”

1. Truncate the Matrix X by removing the k^{th} column:

   ```
   X(:,k)=[];
   ```

2. This also works to remove elements from a vector:

   ```
   x(k)=[];
   ```

3. Increase the Matrix X by adding the k^{th} column of Y:

   ```
   X=[X, Y(:,k)];
   ```

4. And the vector version of adding an element of y to a vector x:

   ```
   x=[x y(k)]
   ```

5. Construct a matrix X by using the odd columns (1-5) of Y:

   ```
   I=1:2:5
   X=Y(:,I);
   ```

6. For projections, if we would like to remove the component along a vector w_j from the column space of matrix P:

   ```
   a=wj'*P/(wj'*wj);
   P=P-wj*a;
   ```