Distinct Feature Learning and Nonlinear Variation Pattern Discovery Using Regularized Autoencoders

by

Phillip Howard

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2016 by the
Graduate Supervisory Committee:

George Runger, Chair
Douglas Montgomery
Pitu Mirchandani
Daniel Apley

ARIZONA STATE UNIVERSITY

May 2016

ABSTRACT

Feature learning and the discovery of nonlinear variation patterns in high-dimensional data is an important task in many problem domains, such as imaging, streaming data from sensors, and manufacturing. This dissertation presents several methods for learning and visualizing nonlinear variation in high-dimensional data. First, an automated method for discovering nonlinear variation patterns using deep learning autoencoders is proposed. The approach provides a functional mapping from a low-dimensional representation to the original spatially-dense data that is both interpretable and efficient with respect to preserving information. Experimental results indicate that deep learning autoencoders outperform manifold learning and principal component analysis in reproducing the original data from the learned variation sources.

A key issue in using autoencoders for nonlinear variation pattern discovery is to encourage the learning of solutions where each feature represents a unique variation source, which we define as distinct features. This problem of learning distinct features is also referred to as disentangling factors of variation in the representation learning literature. The remainder of this dissertation highlights and provides solutions for this important problem.

An alternating autoencoder training method is presented and a new measure motivated by orthogonal loadings in linear models is proposed to quantify feature distinctness in the nonlinear models. Simulated point cloud data and handwritten digit images illustrate that standard training methods for autoencoders consistently mix the true variation sources in the learned low-dimensional representation, whereas the alternating method produces solutions with more distinct patterns.

Finally, a new regularization method for learning distinct nonlinear features using autoencoders is proposed. Motivated in-part by the properties of linear solutions,

a series of learning constraints are implemented via regularization penalties during stochastic gradient descent training. These include the orthogonality of tangent vectors to the manifold, the correlation between learned features, and the distributions of the learned features. This regularized learning approach yields low-dimensional representations which can be better interpreted and used to identify the true sources of variation impacting a high-dimensional feature space. Experimental results demonstrate the effectiveness of this method for nonlinear variation pattern discovery on both simulated and real data sets.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Discovering nonlinear variation patterns is an important problem in many applications involving high-dimensional data. For example, a process engineer may wish to understand the nature of part-to-part variation using profile data of manufactured components where the number of measurements taken over a three-dimensional part is much larger than the number of variation patterns. Similarly, a doctor comparing Medical Resonance Imaging (MRI) brain scans across multiple patients might seek to discover how the topology of the brain changes in patients with Alzheimer's disease. Electrocardiogram (ECG) data is another example where hundreds of time-ordered measurements are used to represent a single heartbeat that is affected by a much smaller set of conditions related to the functioning of the heart. A natural approach to identifying the underlying variation patterns in such high-dimensional data is to reduce the dimensionality in a manner which maximizes the amount of information preserved in the reduced feature space. The resulting features can then be viewed as latent variables representing the underlying variation patterns which have the greatest influenced over the observed data.

One example of dimensionality reduction is Principal Component Analysis (PCA), which is frequently used to extract low-dimensional features from high-dimensional data sources. Because the Principal Component (PC) loadings are the eigenvectors of the data covariance matrix, they can be used to identify uncorrelated linear variation patterns in high-dimensional data by interpreting each PC variable as representing a variation pattern. The ordering of the principal components by the eigenvalues of the covariance matrix further facilitates variation pattern analysis because it conveys

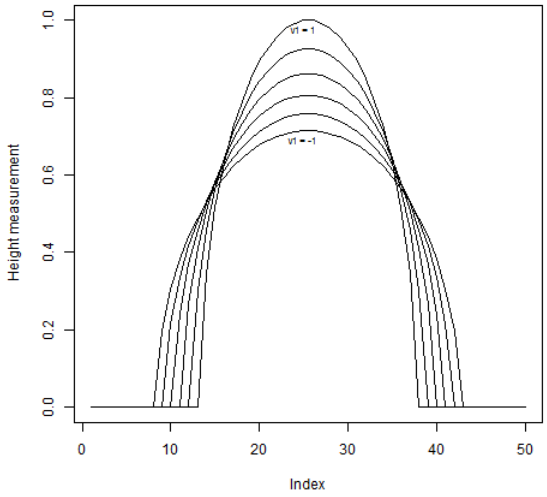

Figure 1.1: 50-Dimensional Gasket Bead Profiles Affected by a Single Nonlinear Variation Pattern

information on the amount of variation in the original data explained by each PC variable. However, PCA is a linear method that fails to accurately characterize the underlying variation patterns when they are nonlinear in nature. This dissertation proposes solutions appropriate for nonlinear variation source discovery which aim to be as useful for identifying distinct variation patterns as PCA solutions are for the linear case.

An example of a nonlinear variation pattern is provided in Figure 1.1, which depicts several 50-dimensional instances of a gasket bead profile affected by a single nonlinear variation determining the amount of flattening along the gasket bead. Each feature in a 50-dimensional instance consists of a single height measurement taken at a discrete location along the length of a part. The single nonlinear variation pattern, denoted as v_1 in the plot, decreases from $v_1 = 1$ to $v_1 = -1$ as the gasket bead is flattened. The variation source in this example is nonlinear in nature due to the

nonlinear functional relationship between v_1 and the 50 discrete height measurements.

1.1 Importance of Analyzing High-Dimensional Spatial Data

Dense spatially-arranged data has become commonplace in areas such as medical imaging, where discovering the nature of physical variations across multiple high-resolution images could help diagnose or better understand a medical condition. Another application area in healthcare is in analyzing high-dimensional genomic sequences to help understand the nature of genetic differences between subjects. Additionally, the prevalence of inter-connected devices in what is commonly referred to as the Internet of Things will result in a plethora of new data from sensors that are arranged over a spatial domain. When such sensor sensor data is also considered over time, the resulting spatio-temporal data yields a high-dimensional feature space influenced by underlying variation patterns that could reflect the overall performance of a system. Due to the dense nature of such data, analytical tools that help facilitate and interpret sources of variation are critical to the extraction of meaningful information.

The ability to discover nonlinear variation patterns in high-dimensional and spatially-arranged data is particularly relevant to modern manufacturing due to the recent advancements in measurement technology which facilitates the extraction of profile data from physical objects. One of the most promising emerging measurement technologies that is broadly applicable in discrete parts manufacturing is noncontact dimensional coordinate metrology using laser and/or vision systems (Shi *et al.*, 2016). Laser measurement can be either point scan (a fine laser beam is projected onto the part as a point and scanned across the part) or line scan (a laser beam is fanned into a plane, projected onto the part as a stripe, and scanned across the part). A single point scan produces a profile, whereas a series of point scans or a line scan produces point cloud data in 3D space. Both represent the surface geometry of the scanned

part. Vision systems take what can be high-magnification images of the part surface, from which parametric (e.g., circles, lines) or nonparametric (e.g., edges, contours) geometric features or general visual characteristics of the part can be extracted with standard image processing algorithms. Laser and/or vision systems generically can be broadly classified as optical coordinate measuring machines (OCMM).

Recent technological advancements in depth-sensing vision systems have also made OCMM more widely available without the need to invest in expensive hardware. 3D cameras such as the Microsoft Kinect extract depth maps in addition to the standard RGB images created by regular cameras. Pictures of an object taken with a 3D camera from multiple angles can be used to fuse information across depth maps and construct a 3D point cloud, which allows widely-available hardware such as the Microsoft Kinect to be used as an inexpensive 3D scanner. Now 3D scanning capabilities are also becoming more widely available as depth-sensing cameras are being integrated into laptops and smartphones. The propagation of hardware for 3D point cloud extraction necessitates analytical tools for automatically analyzing and interpreting such data; these capabilities will allow individuals to better understand the physical world through the lens of consumer electronic devices in ways that were previously not possible.

1.2 Problem Description

The high-dimensional spatial data considered in this work is assumed to have the functional form $\mathbf{x}_n = \mathbf{f}(\mathbf{s}_n) + \mathbf{w}_n$, where $\mathbf{x}_n$ is the n^{th} vector instance in a set of N data observations. A given data instance $\mathbf{x}_n$ consists of a vector existing in an M-dimensional space, where M corresponds to the number of features in the observed data. A set of P unknown variation sources $\mathbf{s}_n = [s_{n,1}, s_{n,2}, \ldots, s_{n,P}]$ give rise to the observed data instance $\mathbf{x}_n$ and are mapped to the M-dimensional space via unknown

nonlinear functions $\mathbf{f}(\mathbf{s}_n) = [f_1(\mathbf{s}_n), f_2(\mathbf{s}_n), \ldots, f_M(\mathbf{s}_n)]^T : \mathbb{R}^P \mapsto \mathbb{R}^M$. The P sources of variation are assumed to be independent, which is a common assumption for variation source discovery problems (Shi *et al.*, 2016). We also assume that independent additive noise $\mathbf{w}_n$ corrupts the n^{th} observed data instance.

Our objective is to identify the P true variation sources $\mathbf{s}_n$ and their nonlinear functions $\mathbf{f}(\ )$ using only the M-dimensional observed data instances $\mathbf{x}_n, n \in \{1, \ldots, N\}$. The motivation for this problem is having the ability to interact with the data in a low-dimensional space where it is much easier to visualize and interpret the variation patterns. For example, the original data instances can often exist in $M > 1000$ dimensions when working with images consisting of thousands of pixel intensities. This high-dimensional representation cannot be easily visualized in a manner which facilitates comparisons across the set of N data instances, whereas projecting the data to a low-dimensional space (e.g., $P = 2$ dimensions) allows the N data instances to be simultaneously visualized in a single scatter plot. Additionally, identifying both $\mathbf{s}_n$ and $\mathbf{f}(\mathbf{s}_n)$ provides the ability to understand the range over which each nonlinear variation pattern occurs as well as the functional mapping of the pattern $\mathbf{s}_n$ to the observed feature space of $\mathbf{x}_n$. We use the learned $\mathbf{s}_n$ and $\mathbf{f}(\mathbf{s}_n)$ together to achieve our goal of interactively visualizing the variation patterns by changing the value of $\mathbf{s}_n$ over its observed range and then projecting back to the observed feature space via $\mathbf{f}(\mathbf{s}_n)$.

1.3 Analytical Solutions Using Autoencoders

This dissertation explores analytical solutions for identifying and visualizing nonlinear variation patterns in high-dimensional spatial data. In particular, the solutions presented in this dissertation employ autoencoders (Kramer, 1991; Hinton and Salakhutdinov, 2006; Bengio *et al.*, 2013), which are sometimes referred to as autoas-

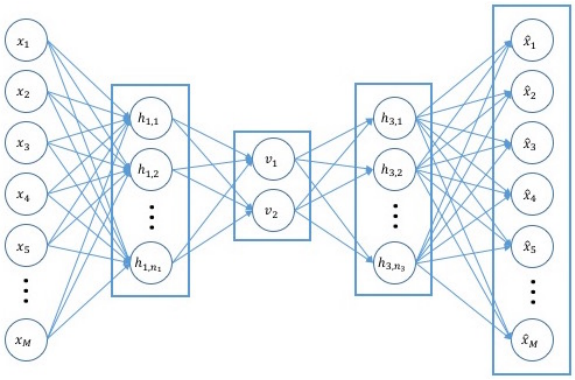

Figure 1.2: Example of an Autoencoder Network Structure

sociative neural networks or nonlinear principal component analysis. An autoencoder is a neural network-type model which is trained to reproduce its input at its output layer after passing through several hidden layers of artificial neurons. An example of the basic network structure with three hidden layers is provided in figure 1.2. The middle layer of the network, which is sometimes referred to as the bottleneck layer or encoding layer, has many fewer hidden units than the dimensionality of the input. This forces information compression as the model attempts to learn the best possible reconstruction of the input data given only the information encoded into the constrained middle layer.

Autoencoders are particularly well-suited for variation pattern discovery because they simultaneously learn both latent variables representing the nonlinear variation patterns $\mathbf{s}_n$ and the nonlinear functions $\mathbf{f}(\mathbf{s}_n)$ which map the latent variables back to the observed feature space. This provides the ability to easily visualize the learned patterns in the original feature space and this leads to the capability to interpret the variation sources for various values of the latent variables.

There has been a resurgence of interest in neural network-type models such as autoencoders in recent years due to the substantial advancements made in 'deep learning' research (LeCun *et al.*, 2015). Deep learning is often used to describe neural

network models in which many hidden layers are used, thereby resulting in deep network architectures. These deep networks were once considered unpractical to train due to the large number of parameters and the tendency for gradients to vanish as they pass through multiple layers of nonlinear activation functions. However, alternative training methods introduced recently have substantially increased the speed at which deep networks can be learned, allowing them to efficiently achieve state-of-the-art performance on many benchmark problems in machine learning LeCun *et al.* (2015).

Deep network architectures allow the learning of features representing multiple levels of abstraction of the data, where each level in the network consists of features learned from the previous layer's features. The deeper layers in the model are farther separated from the input data, allowing the layers closer to the inputs to serve as feature detectors which can then be considered collectively at a deeper level in the network where higher-level information is extracted. In the autoencoder model depicted in figure 1.2, the small number of nodes in the middle layer can be interpreted as the type of abstract learned feature that makes deep learning models successful.

1.4 Distinct Feature Learning

The existing research on deep learning has been primarily concerned with simply learning abstract features internally within the model for the purpose of improving the accuracy of the model output. The research presented in this dissertation advances the current state of knowledge by introducing techniques that encourage the learned features to represent unique variation patterns, thereby expanding the applicability of deep learning to a range of problems in which it is important to be able to understand what has been learned by the model. We refer to features representing unique variation patterns as 'distinct' features, and our task of learning such features is sometimes referred to as disentangling factors of variation in the representation

learning literature (Bengio *et al.*, 2013). The learning of distinct features is a new field of research with relatively little prior work pertaining to autoencoder models.

Distinct feature learning is an important problem because often the features learned by an autoencoder model represent some combination of multiple nonlinear variation patterns rather than the true distinct and unique nonlinear variation sources impacting the data. This occurs because existing training methods only consider minimizing the sum of squared error of the model's reconstruction of the data, causing the model's representation of the variation sources to all align early in training with a single pattern that accounts for the greatest amount of variation. The resulting models are not interpretable and fail at the task of visualizing the true nonlinear sources of variation. Thus, learning the distinct patterns is a critical challenge that must be addressed in order to make the identified variation sources interpretable when visualized.

Consider the gasket bead example depicted in Figure 1.1, and suppose that there is a second variation source which shifts the gasket bead from left to right in the 50-dimensional space (i.e. controlling the index at which the nonlinear portion of the part begins). An autoencoder with two bottleneck nodes that is trained only to minimize the sum of squared error in reconstructing the data instances could learn a pair of features where the shifting variation pattern is characterized by both features, with one feature characterizing bead flattening and shifting in one direction while the other characterizes bead elongation and shifting in the opposite direction. This obscures the true sources of variation when interactively visualizing the learned features and could make the identification of the true variation patterns impossible as more features are added to the model.

1.5 Dissertation Outline

Chapter 2 provides background material on several concepts discussed in later chapters of the dissertation. An overview of autoencoders is provided along with a discussion of the typical method used to train such models. The Restricted Boltzmann Machine and its relationship to the unsupervised pre-training method commonly used for deep learning is also detailed. Finally, a discussion of the relationship between blind source separation, independent features, and orthogonal vectors is provided.

Chapter 3 of this work presents a new method for learning nonlinear variation patterns in two-dimensional data by reindexing the data in a manner which transforms nonlinear patterns into linear patterns, thereby allowing linear methods such as PCA to be applied. A second contribution of Chapter 3 is the application of recent advances in deep learning to the nonlinear variation pattern discovery problem through the use of pre-training techniques for autoencoders that have many hidden layers. We show that these deep autoencoders can effectively discover and visualize the nonlinear patterns while also generalizing well to three-dimensional point cloud data. The performance of both methods are compared to manifold learning and regular PCA.

Chapter 4 addresses the challenge of ensuring that the learned variation patterns are distinct when using autoencoders for variation source discovery. To overcome this problem, a new method is presented for sequentially training the model parameters in a manner that guides the model towards a solution where the extracted patterns are distinct and interpretable. Chapter 4 also introduces a numerical measure for the degree to which the learned patterns are distinct and separate, which facilitates analytical comparisons of model performance on the basis of both quality of fit and interpretability. The measure introduced in Chapter 4 can be used to compare alternative approaches in this new field of research which we refer to as distinct feature

learning. The performance of our sequential learning method is compared to regular training methods across several examples involving image and 3D point cloud data.

Chapter 5 introduces a new regularized learning method for learning distinct features using autoencoders. In contrast to the sequential learning method presented in Chapter 5, this method imposes several constraints on the model during training to encourage the learning of solutions where the nonlinear features represent unique variation sources. These constraints include an uncorrelated relationship between the learned features and orthogonal tangent vectors on the manifold learned by an ANN, which is analogous to properties of a PCA solution. Finally, Chapter 6 summarizes the contributions of this dissertation and discusses future work on the topic of learning distinct nonlinear features using autoencoders.

Chapter 2

BACKGROUND OF NEURAL NETWORKS, DEEP LEARNING, AND SOURCE SEPARATION

This chapter provides background material on several concepts discussed later in the dissertation. An overview of autoencoders is provided along with a discussion of the typical method used to train such models. The Restricted Boltzmann Machine and its relationship to the unsupervised pre-training method commonly used for deep learning is also detailed. Finally, a discussion of the relationship between blind source separation, independent features, and orthogonal vectors is provided. In additional to the background material provided here, Chapters 3, 4, and 5 also provide a brief discussion of previous work relevant to the respective chapters.

2.1 Autoencoder Models

An ANN model is a network of hidden nodes (also referred to as neurons) which connect the inputs of the model to its outputs through a series of layers. An example of a simple ANN network structure is provided in Figure 2.1. The inputs to the model consist of the elements of a given data vector $\mathbf{x}_n = [x_{n,1}, x_{n,2}, \ldots, x_{n,M}]$ and the output layer has the same dimensionality as the input (M). The objective in fitting an ANN model is to reproduce the inputs to the model at the output layer after forcing information compression within the network.

Each layer in the network is connected to the nodes in the previous layer through a set of weights, which are depicted as arcs in Figure 2.1. The i^{th} node of layer $k-1$ is connected to the j^{th} node of layer k by weight $w_{i,j}^{(k)}$ where layer 0 corresponds to the input. Each hidden node has a nonlinear activation function which determines how

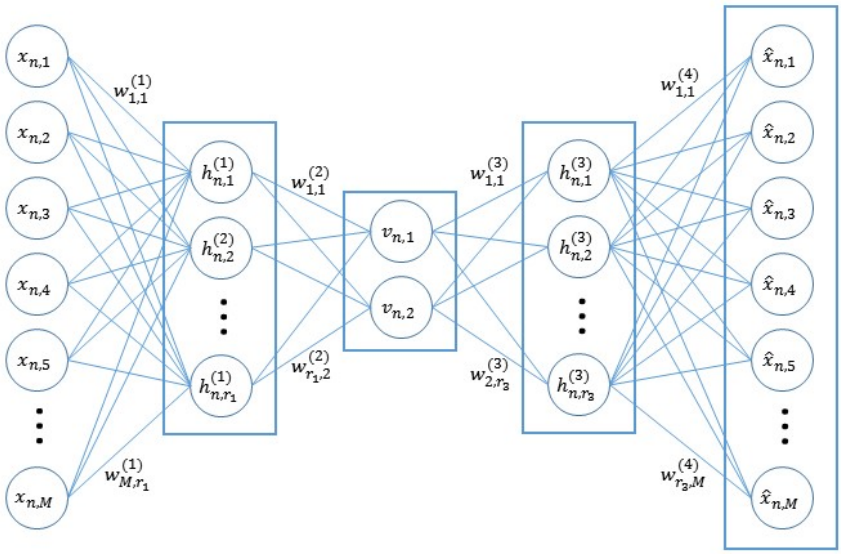

Figure 2.1: Network Architecture for an ANN Model With $P = 2$ Components

the input from the previous layer is functionally related to the next layer. Letting $h^{(k)}_{n,j}$ denote the j^{th} hidden node in layer k and evaluated using data instance n, the value of $h^{(k)}_{n,j}$ determined by the sigmoid activation function is

$$h^{(k)}_{n,j} = \frac{1}{1 + \exp\left(w^{(k)}_{0,j} + \sum_{i=1}^{r_{k-1}} w^{(k)}_{i,j} h^{(k-1)}_{n,i}\right)} \tag{2.1}$$

where in Equation 2.1, r_k denotes the number of hidden nodes in the k^{th} layer of the network and $w^{(k)}_{0,j}$ is a bias weight for the j^{th} node of layer k. Another commonly used activation function is the linear activation. For a given layer k with linear activations, $h^{(k)}_{n,j}$ is defined as

$$h^{(k)}_{n,j} = w^{(k)}_{0,j} + \sum_{i=1}^{r_{k-1}} w^{(k)}_{i,j} h^{(k-1)}_{n,i} \tag{2.2}$$

The weights of an ANN model are learned by optimizing an objective function using methods such as stochastic gradient descent. A common objective function for ANNs is the Sum of Squared Error (SSE), which measures how similar the output vector of the network is to the input data vector. Letting $\hat{x}_{n,m}$ denote the m^{th} output

node of an ANN produced by input vector $\mathbf{x}_n$, the SSE evaluated using weight set $\mathbf{W}$ is denoted $J_E(\mathbf{W})$ and defined as

$$J_E(\mathbf{W}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}(x_{n,m} - \hat{x}_{n,m})^2 \tag{2.3}$$

The SSE encourages an ANN to reconstruct the original input vector as closely as possible at its output layer. Dimensionality reduction is achieved by the middle layer of the network, commonly referred to as the bottleneck layer, having a dimensionality $P < M$ to force information compression. The P nodes in the bottleneck layer are denoted $\mathbf{v}_n = [v_{n,1}, v_{n,2}, \ldots, v_{n,P}]$ and represent the model's learned representation of the true unknown variation sources $\mathbf{s}_n$. Figure 2.1 depicts an ANN having $P = 2$ bottleneck nodes in the bottleneck layer. The portion of the network consisting of the bottleneck layer through the output layer is referred to as the decoder and denoted $\hat{\mathbf{f}}(\mathbf{v}_n)$ because it represents the model's approximation of the true nonlinear functions $\mathbf{f}(\mathbf{s}_n)$ governing the data.

The SSE of reconstruction is a desirable criterion to minimize during ANN training because it encourages the model to learn accurate representations of the original data. However, the flexibility in modeling nonlinear relationships provided by an ANN allows many different functional mappings to and from the bottleneck layer to be learned which each produce similar levels of SSE efficiency. If only the SSE of reconstruction is optimized during training, an ANN's learned representation of the true variation sources $\mathbf{s}_n$ and nonlinear functions $\mathbf{f}(\ )$ can therefore be inaccurate and difficult to interpret. This could be due to multiple bottleneck nodes aligning with the same variation source early in training when the gradients corresponding to the most influential variation pattern are largest (Kramer, 1991), leading to solutions where each of the nonlinear features represents a combination of the true variation sources. In Chapters 4 and 5, we present alternative objective functions for ANN

training which avoid this deficiency.

2.2 Deep Learning

As discussed in Chapter 1, the advent of new training methods in recent years has led to renewed interest in neural network models. Unsupervised pre-training is one such advancement which has enabled the learning of deep network structures. The following subsections provide background on an important component of unsupervised pre-training, the Restricted Boltzmann Machine, and discuss the role that unsupervised pre-training has in deep learning. Our application of unsupervised pre-training to deep autoencoders follows in Chapter 3.

2.2.1 Boltzmann Learning

This section uses the notation and several examples originally presented by Duda et al. (Duda *et al.*, 2012). An illustration of a Boltzmann machine is provided in figure 2.2. The network is comprised of a set of visible (input) units, collectively denoted as $\boldsymbol{\alpha}$, a set of hidden units denoted by $\boldsymbol{\beta}$, and weights w_{ij} connecting visible unit i to hidden unit j. In this example, all of the hidden units are fully connected to the visible units and there are no connections existing between hidden units. This type of network structure is referred to as a Restricted Boltzmann Machine (RBM) because the hidden units are restricted to only have connections to the visible units in the layer below. I will focus on RBMs because they are used in unsupervised pre-training of neural networks, but Boltzmann machines in general do not have this restriction and can have arbitrary connections existing between hidden units.

Boltzmann machines are governed by an energy function E which, for a network consisting of N total nodes (including both visible and hidden), is defined as

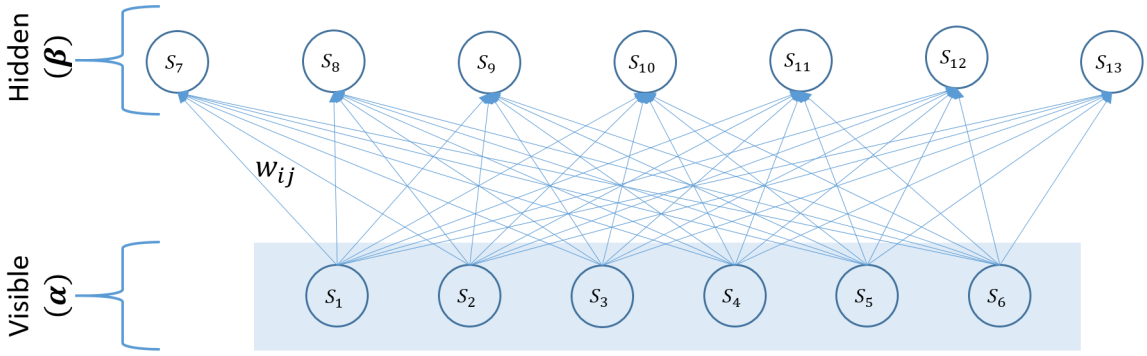

Figure 2.2: An Example of a Restricted Boltzmann Machine With Visible (Input) Units $\boldsymbol{\alpha}$ and Hidden Units $\boldsymbol{\beta}$. the Arcs Represent Weights Connecting Visible Nodes to Hidden Nodes, Where w_{ij} Denotes the Weight Connecting Visible Node i to Hidden Node j.

$$E = -\frac{1}{2}\sum_{i,j=1}^{N} w_{ij} s_i s_j \tag{2.4}$$

In the above equation, s_i and s_j denote the state of nodes i and j in the network (respectively). The energy function is inspired by applications of Boltzmann learning in physics. For example, suppose each of the nodes s_i in the network represents a physical magnet that can take on the value of $+1$ or -1 corresponding to the magnet's polarity (temporarily ignore the distinction between visible and hidden nodes for this example). Given that the weights of the network $w_{i,j}$ are known, a typical optimization task in Boltzmann learning is to find the values of the states s_i for $i \in \{1, \ldots, N\}$ which minimizes the total energy of the system defined by equation 2.4. This lowest-energy state corresponds to the most stable arrangement of the system (e.g. the most stable arrangement of a set of magnets).

Simulated annealing is a method inspired by physics which is frequently used to perform this task of minimizing the energy of a system. Physical annealing is a process where a system his heated in order to impose randomness on the components in the system (such as atoms in an alloy), thereby allowing each of the elements to

temporarily assume values which are not necessarily optimal for minimizing the overall energy the function. As the temperature is gradually lowered, less randomness is imposed and individual elements are more likely to settle into states that are (locally) optimal given the current configuration of the system. This randomness allows the system to explore a vast area of energy landscape early in the annealing process when the temperature is high, thereby avoiding poor local minima to which the system is likely to converge when the states are only randomly initiated.

Returning to the original example of an RBM depicted in figure 2.2, let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ denote the visible and hidden node states for a particular configuration of the network which has a corresponding energy $E_{\boldsymbol{\alpha\beta}}$. The probability of the system being in this configuration at a given temperature T is defined as

$$\begin{aligned} P(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{e^{-E_{\boldsymbol{\alpha\beta}}/T}}{Z} \\ \text{where} \quad Z &= \sum_{\boldsymbol{\alpha}', \boldsymbol{\beta}'} e^{-E_{\boldsymbol{\alpha}', \boldsymbol{\beta}'}/T} \end{aligned} \tag{2.5}$$

The numerator in equation 2.5 is referred to as the Boltzmann factor and the denominator Z is a partition function which ensures that the value is a probability. Equation 2.5 reflects the fact that there are exponentially decreasing number of possible configurations of the system as the energy increases, and that at higher temperatures (i.e. more imposed randomness), there is more energy available and therefore a higher probability of arriving at a high energy state (Duda *et al.*, 2012). Given this joint probability of the configuration $\boldsymbol{\alpha}, \boldsymbol{\beta}$ of the visible and hidden nodes, we can analogously define the marginal probability of observing the visible node configuration $\boldsymbol{\alpha}$ as

$$P(\boldsymbol{\alpha}) = \frac{\sum_{\boldsymbol{\beta}} e^{-E_{\boldsymbol{\alpha\beta}}/T}}{Z} \tag{2.6}$$

Equation 2.6 describes the probability of observing the configuration $\boldsymbol{\alpha}$ of the visible units when we randomly simulate the states of the network through annealing. Suppose that we have a desired probability $Q(\boldsymbol{\alpha})$ for all possible configurations of the visible units; $Q(\boldsymbol{\alpha})$ could be defined, for example, by a set of training data which serve as the input (visible) units for the network when it is used. The learning task we are concerned with is how to set the weight connections w_{ij} in the network so that the actual probabilities of all visible patterns $P(\boldsymbol{\alpha})$ is as close to our target probabilities $Q(\boldsymbol{\alpha})$ as possible when we perform simulated annealing on the network.

The objective function that is typically used for this learning task is the Kullback-Leibler divergence, which measures the distance between the network's actual probabilities $P(\boldsymbol{\alpha})$ and our target probabilities $Q(\boldsymbol{\alpha})$. It is defined as

$$\begin{aligned} D_{KL}(Q(\boldsymbol{\alpha}), P(\boldsymbol{\alpha})) &= \sum_{\boldsymbol{\alpha}} Q(\boldsymbol{\alpha}) \log \frac{Q(\boldsymbol{\alpha})}{P(\boldsymbol{\alpha})} \\ &= \sum_{\boldsymbol{\alpha}} Q(\boldsymbol{\alpha}) \log Q(\boldsymbol{\alpha}) - \sum_{\boldsymbol{\alpha}} Q(\boldsymbol{\alpha}) \log P(\boldsymbol{\alpha}) \end{aligned} \tag{2.7}$$

Gradient descent is performed to iteratively update the weights according to the partial derivatives of equation 2.7 with respect to each of the parameters. Letting $s_i(\boldsymbol{\alpha}, \boldsymbol{\beta})$ and $s_j(\boldsymbol{\alpha}, \boldsymbol{\beta})$ denote the states of nodes i and j in the configuration defined by sets $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the weight update rule for $w_{i,j}$ is (Duda *et al.*, 2012):

$$\begin{aligned} \Delta w_{ij} &= \eta \frac{\partial D_{K,L}}{\partial w_{i,j}} \\ &= \sum_{\boldsymbol{\alpha}} \frac{Q(\boldsymbol{\alpha})}{P(\boldsymbol{\alpha})} \frac{\partial P(\boldsymbol{\alpha})}{\partial w_{ij}} \\ &= \frac{\eta}{T} \left(\sum_{\boldsymbol{\alpha}\boldsymbol{\beta}} Q(\boldsymbol{\alpha}) P(\boldsymbol{\beta}|\boldsymbol{\alpha}) s_i(\boldsymbol{\alpha}\boldsymbol{\beta}) s_j(\boldsymbol{\alpha}\boldsymbol{\beta}) - E[s_i s_j] \right) \\ &= \frac{\eta}{T} \left(E_Q[s_i s_j]_{clamped} - E[s_i s_j] \right) \end{aligned} \tag{2.8}$$

In equation 2.8, $E_Q[s_i s_j]_{clamped}$ denotes an expectation of the product of states s_i

and s_j when the visible units are clamped (or fixed) to configuration $\boldsymbol{\alpha}$, where the expectation is averaged over all configurations $\boldsymbol{\alpha}$ according to their target probabilities $Q(\boldsymbol{\alpha})$ in our training set. The second term on the right side of equation 2.8 is an expectation of the product of states s_i and s_j without clamping any particular training example on the input node states, and therefore represents the expected product of the state values when values for all of the network nodes are simulated during annealing. The weight update rule in equation 2.8 will be used when explaining the contrastive divergence method for unsupervised neural networks in the next section.

2.2.2 Product of Experts and Unsupervised Pre-Training

A motivation for the use of RBMs in unsupervised pre-training of neural networks is the notion that a Product of Experts (PoE) model can learn a good representation of high-dimensional input data. Similar to mixture models, PoE models combine multiple simpler distributions together in order to form a good approximation of a much more complicated high-dimensional distribution. The individual distributions are combined by multiplying them together and then renormalizing the product to ensure that it is a proper distribution. PoE models can provide more efficient approximations of high-dimensional distributions than mixture models and are often very powerful when the individual 'expert' distributions contain latent variables which model some subset of the high-dimensional feature space (Hinton, 2002).

PoE models are a natural fit for initiating hidden nodes in a neural network because the latent variables learned by each of the individual expert distributions convey some piece of information about the input feature space which is useful for modeling the entire distribution of observed data. If a PoE model is used to learn a set of hidden nodes directly connected to the input data of a neural network, subsequent hidden layers in the network can combine these latent variables to learn more abstract

representations of the data which are useful for tasks such as discrimination and compression. The model parameters learned by a PoE therefore represent an intuitive starting point for the weights in deep neural networks.

Restricted Boltzmann Machines like the one illustrated in figure 2.2 represent PoE models with one expert per hidden unit (Hinton, 2002). To train the weights of an RBM, the typical optimization criteria is to maximize the likelihood of the observed training data given the PoE model. If the PoE consists of n experts each with an associated parameter vector $\boldsymbol{\theta}_n$ (for example, the parameter vector for expert s_7 in figure 2.2 is $\boldsymbol{\theta_7} = [w_{17}, w_{27}, \ldots, w_{67}]$), the probability of observing a data vector $\boldsymbol{\alpha}$ under the model is

$$p(\boldsymbol{\alpha}|\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n) = \frac{\prod_m f_m(\boldsymbol{\alpha}|\boldsymbol{\theta}_m)}{\sum_{\boldsymbol{\alpha'}} \prod_m f_m(\boldsymbol{\alpha'}|\boldsymbol{\theta}_m)} \tag{2.9}$$

In equation 2.9, $f_m(\boldsymbol{\alpha}|\boldsymbol{\theta}_m)$ represents the probability of observing data vector $\boldsymbol{\alpha}$ under expert m and $\boldsymbol{\alpha'}$ is used to index across all of the possible data vectors in the feature space. Maximizing the likelihood defined in equation 2.9 averaged across the distribution of observed data is equivalent to minimizing the Kullback-Leibler divergence between the observed data distribution over the inputs $Q(\boldsymbol{\alpha})$ and the actual distribution over the inputs produced by the model over prolonged simulation, denoted as $P(\boldsymbol{\alpha})$ (Hinton, 2002). Thus, learning the parameters for a PoE model is equivalent to minimizing equation 2.7 presented in the previous section and has the same weight update rule as defined in equation 2.8.

We can obtain the expected values $E_Q[s_i s_j]_{clamped}$ and $E[s_i s_j]$ to update the RBM weights as described in equation 2.8 through the use of Gibbs Sampling. An illustration of Gibbs Sampling is provided in figure 2.3 (Hinton, 2002). In this figure, $< s_i, s_j >_t$ denotes the product of states s_i and s_j after the t iterations of sampling. The procedure begins by placing a data vector on the visible units and then allowing

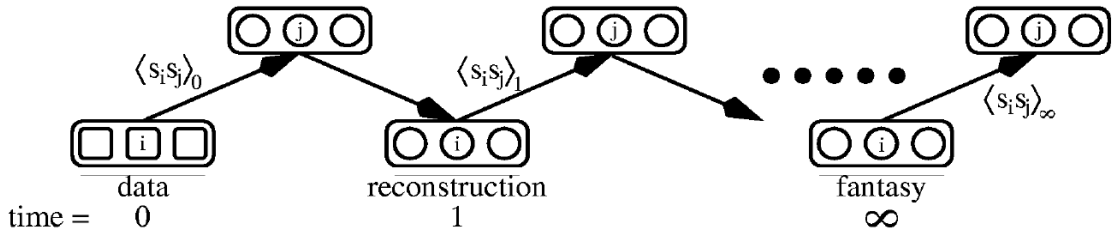

Figure 2.3: Example of Gibbs Sampling (Hinton, 2002)

the hidden units to update their values based on the weights of the network. The resulting hidden values are then used to create a reconstruction of the input by using the same weight connections, which lead to a new set of corresponding hidden node states. If this process is repeated an infinite number of times, the product of the states $< s_i, s_j >_\infty$ represents our desired expectation $E[s_i s_j]$, which is the expected value of the state products when the model is 'unclamped' from the training data and producing states purely based on the distribution learned by the model. The state value product $< s_i, s_j >_0$ produced when a data vector is still clamped to the inputs provides our other necessary expectation $E_Q[s_i s_j]_{clamped}$. Thus, the maximum likelihood weight update rule for an RBM with learning rate η is

$$\Delta w_{ij} = \eta(< s_i, s_j >_0 - < s_i, s_j >_\infty) \tag{2.10}$$

It is impractical to compute an infinite number of Gibbs Sampling steps, so the contrastive divergence method of weight updates simplifies this by only performing 1 round of sampling and setting the weight update equal to

$$\Delta w_{ij} = \eta(< s_i, s_j >_0 - < s_i, s_j >_1) \tag{2.11}$$

This weight update rule is no longer following the gradient of the likelihood of the data, but is instead following the gradient of a function called the contrastive divergence. Specifically, the contrastive divergence is the difference between the Kullback-

Leibler divergence in equation 2.7 when it is evaluated with $E_Q[s_i s_j]_{clamped} = < s_i, s_j >_0$ and when it is evaluated with $E_Q[s_i s_j]_{clamped} = < s_i, s_j >_1$ (Hinton, 2002). This simplification makes learning the weights analytically tractable and performs well in practice.

Contrastive divergence is used to pre-train each layer of a deep neural network sequentially, starting with the first hidden layer connected directly to the model inputs. Once the first hidden layer has been trained, the second hidden layer is trained by treating the outputs of the first hidden layer evaluated on the training data as inputs for the second hidden layer, and so forth until each hidden layer is trained. The entire network is then fine-tuned with standard backpropogation methods to minimize an objective function, such as the sum of squared errors evaluated at the output of the network.

2.2.3 Role of Restricted Boltzmann Machines in Improving Deep Networks

Since the introduction of RBMs as a form of unsupervised pre-training of deep neural networks, there has been much discussion in the machine learning literature regarding how unsupervised pre-training aids in the learning of deep architectures. Without pre-training, the parameters of a network are typically initiated randomly and then gradient descent is performed until the weights converge to a local minimum in the error surface. Empirical results have shown that performing hundreds of different random initializations of the parameters will each lead to unique local minima (Erhan *et al.*, 2009a), suggesting that the error surface is complex and that alternative initialization methods for the parameters are needed to find good regions for gradient-based searches.

A discussion of several theories previously suggested in the literature for why unsupervised pre-training aids deep learning was presented in (Erhan *et al.*, 2010).

The objective function used in supervised learning contains many local minima, and the issue becomes more pronounced as additional layers are added to a network. For deep networks, randomly initialized parameters will fall within a basin of attraction to a poor local minima with very high probability. Unsupervised pre-training avoids this by initiating the parameters in a basin of attraction to minima containing better generalization performance. Experimental results presented by the authors support the hypothesis that unsupervised pre-training traps the parameters in a basin of attraction to solutions that are useful for unsupervised learning, thereby avoiding overfitting that can occur when only supervised learning is used.

A discussion of the difficulties involved in learning deep architectures as well as the effect of unsupervised pre-training was provided in (Erhan *et al.*, 2009b). The authors show how pre-training results in both a better optimization procedure and leads to a solution with better generalization performance. They also suggest that pre-training acts as a form of regularization by restricting the starting points for optimization to a manifold that is dependent on the training data. Their experiments also show that the effect of pre-training increases as the size of the training data set decreases. They also show that pre-training does not help (and in fact leads to worse performance) when there are few layers in the network, and that pre-training has the greatest effect on the earlier layers in the network (i.e. those closest to the input).

The relationship between activation functions used in a network and poor performance of randomly initiated weights in deep architectures has been explored (Glorot and Bengio, 2010). Nonlinear activation functions such as the logistic sigmoid unit are shown to be a poor fit for random initialization in deep networks, resulting in slower convergence to poorer local minima in the error surface. Alternative activation functions such as rectified linear units have shown to produce faster convergence than the standard sigmoid activations (Dahl *et al.*, 2013), suggesting that the ad-

vantage provided by pre-training depends in part on the configuration of the hidden node activations.

Based on aforementioned findings and the PoE motivation for unsupervised pre-training, we conclude that RBMs primarily improve deep network architectures by learning a starting point for the parameters which provides a good probabilistic representation of the observed training data. By starting gradient-based supervised learning in these regions of the error surface which are useful for unsupervised learning, the model avoids getting stuck in poor local minima where the parameters are likely to start near if only random initialization is used. This is particularly important for deep network architectures because the probability of randomly initializing the weights in the basin of attraction to a poor local minima increases as more layers are added to the network.

2.3 Relationship Between Independence, Orthogonality, and Source Separation

Chapters 4 and 5 of this dissertation discuss the independence of learned features, orthogonality of tangent vectors on a manifold, and the blind source separation problem. This section provides a brief background on these topics and their relationship to PCA, factor analysis, and independent component analysis in order to clarify concepts used later in Chapters 4 and 5.

2.3.1 Principal Component Analysis and Factor Analysis

A common method for generating a set of uncorrelated variables is PCA. Principal components are found by calculating the eigenvectors and eigenvalues of the covariance matrix computed over a data set. The first PC variable characterizes the greatest amount of variation in the original data and is defined by the eigenvector of the covariance matrix having the largest corresponding eigenvalue (Duda *et al.*,

2012). The eigenvectors are sometimes referred to as the PC loadings and provide a functional mapping from the original data space to the PC variables. The rest of the PC variables are found by calculating their loadings from the remaining eigenvectors ordered according to their corresponding eigenvalues, and each one can be interpreted as capturing the greatest amount of variability in the data subject to the constraint of having a loading vector which is orthogonal to the loadings of all previously-defined PC variables.

PCA is the unique linear solution which has both orthogonal loadings in the original data space and uncorrelated variables in the new data space to which the data is transformed. Despite producing a set of uncorrelated latent variables, the lack of correlation provided by PCA is only a necessary and not a sufficient condition of statistical independence. Two variables X and Y are uncorrelated if their covariance $Cov(X,Y) = E[XY] - E[X]E[Y] = 0$, whereas they are statistically independent if and only if their joint pdf factors into a product of their marginal pdfs, i.e. $f_{X,Y}(x,y) = f_X(x)f_Y(y)$, or equivalently, if $f_{Y|X}(y|x) = f_Y(y)$. A simple example is if we define a standard normal variable $X \sim N(0,1)$ and let $Y = X^2$. We have

$$\begin{aligned} Cov(X,Y) &= E[XY] - E[X]E[Y] \\ &= E[X^3] - 0 \\ &= 0 \end{aligned} \tag{2.12}$$

X and Y are therefore uncorrelated. However, Y is completely dependent on X by definition, so it is not true that $f_{Y|X}(y|x) = f_Y(y)$.

The terms 'orthogonal' and 'uncorrelated' are often used interchangeably when discussing principal components because PCA provides the unique linear solution with both orthogonal loadings and uncorrelated variables. However, the condition of orthogonality applies to the vector of PC loadings whereas the vectors of PC variables

are uncorrelated. Furthermore, orthogonality and uncorrelated are not equivalent concepts. If $\mathbf{x}$ and $\mathbf{y}$ are vector observations of the random variables x and y, then $\mathbf{x}'\mathbf{y} = 0$ is a necessary and sufficient condition for orthogonality. However, vectors $\mathbf{x}$ and $\mathbf{y}$ are uncorrelated if and only if $(\mathbf{x} - \bar{x}\mathbf{1})'(\mathbf{y} - \bar{y}\mathbf{1}) = 0$ where $\mathbf{1}$ is a vector of ones and $\bar{x}, \bar{y}$ represent the means of $\mathbf{x}$ and $\mathbf{y}$ (Rodgers *et al.*, 1984). Orthogonality implies that the vectors themselves are perpendicular, whereas uncorrelated indicates that the vectors are perpendicular after they have been centered (i.e. subtracted by their means).

Another approach commonly used for identifying latent variables is factor analysis. In factor analysis, an observed p-dimensional data vector $\mathbf{x}$ is assumed to have the functional form

$$\mathbf{x} = \mathbf{A}\mathbf{f} + \mathbf{e} \tag{2.13}$$

where the vector $\mathbf{f}$ consists of m unknown factors, $\mathbf{A}$ is a $p \times m$ matrix of unknown factor loadings, and $\mathbf{e}$ is a vector of unobserved error terms which are assumed to be uncorrelated and have mean zero. Factor analysis therefore assumes a model structure to achieve a dimensionality reduction from p to m, which differs from PCA in that PCA has no explicit model structure (Jolliffe, 2002).

Standard assumptions of factor analysis include centering of the data, i.e. $E[\mathbf{x}] = \mathbf{0}$, and also that $E[\mathbf{f}] = \mathbf{0}$. Additionally, it is often assumed that $E[\mathbf{f}\mathbf{f}'] = \mathbf{I}_m$ where $\mathbf{I}_m$ is the m-dimensional identity matrix, which implies that the factors are orthogonal as in principal component analysis. However, this assumption is sometimes dropped when performing factor analysis in order to allow for the discovery of oblique factors which can be more interesting for interpreting the solution. Both the factors $\mathbf{f}$ and the loadings $\mathbf{A}$ are unknown and have to be estimated, which implies that solutions are highly nonunique. Different estimation techniques used in factor analysis can

therefore yield different results.

2.3.2 Independent Component Analysis and Blind Source Separation

While both PCA and factor analysis can be used to obtain a set of uncorrelated latent variables, neither of these approaches produce solutions where the derived latent variables are guaranteed to be independent. An alternative method which does produce statistically independent latent varietals is Independent Component Analysis (ICA). The objective of ICA is to identify a linear transformation of the data which produces latent variables that are as independent as possible. This is performed by optimizing some measure of the independence of the latent variables, such as the joint entropy (Duda *et al.*, 2012).

A motivation for ICA is the blind source separation problem, which assumes that there are d independent source signals present in a given instance, denoted as x_i for $i = 1, \ldots, d$ and collectively in vector form as $\mathbf{x}$. The problem assumes that only a k-dimensional vector $\mathbf{s}$ is observed where $\mathbf{s} = \mathbf{A}\mathbf{x}$. The matrix $\mathbf{A}$ is an unknown $k \times d$ mixing matrix which obscures the true signals $\mathbf{x}$ in the observed data. The objective in blind source separation is to identify the true source signals $\mathbf{x}$ using only the observed data $\mathbf{s}$. A classic example is where the k sources $\mathbf{s}$ represent microphones placed throughout a crowded room with multiple speakers; each individual speaker represents one of the x_i original sources that we wish to uncover.

For the linear case in which each of the observed signals $\mathbf{s}$ represents a linear combination of the source signals $\mathbf{x}$, it can be shown that the blind source separation problem and the ICA problem are equivalent (Comon, 1994). ICA is therefore used to perform blind source separation when the observed data is a linear function of the true sources because finding a linear transformation of the $\mathbf{s}$ which yields independent components is equivalent to identifying the original $\mathbf{x}$ sources.

Unfortunately this equivalence between ICA and blind source separation does not extend to the nonlinear case (Jutten and Karhunen, 2003). Specifically, consider a nonlinear mixture model in which the observed signals have the form $\mathbf{x} = \mathbf{F}(\mathbf{s})$, where $\mathbf{F}$ is an unknown mixing function with k components which provide a nonlinear mapping of the d source signals $\mathbf{s}$ to the k-dimensional space of $\mathbf{x}$. The nonlinear blind source separation problem is to identify the original source signals using only the observed data $\mathbf{x}$. This differs fundamentally from the nonlinear ICA problem, which is to find a functional mapping $\mathbf{G}()$ of $\mathbf{x}$ to a new set of latent variables $\mathbf{y} = \mathbf{G}(\mathbf{x})$ such that the resulting $\mathbf{y}$ are as statistically independent as possible. One reason for the lack of equivalence is that while there exists only one unique solution to the nonlinear blind source separation problem, there are many nonunique solutions to nonlinear ICA because any functions $f(x)$ and $f(y)$ of two independent random variables x and y will also be independent (Jutten and Karhunen, 2003).

Another major challenge in nonlinear blind source separation is that the learned latent variables $\mathbf{y}$ can have the original source signals $\mathbf{s}$ completely mixed amongst themselves and yet still be statistically independent. An example of this effect with $k = d = 2$ source signals and observed variables was provided in (Jutten and Karhunen, 2003). Suppose the first source signal s_1 follows a Rayleigh distribution with pdf $f_{s_1}(s_1) = s_1 e^{-s_1^2/2}$ and the second source signal is independent of s_1 with uniform pdf $f_{s_2}(s_2) = \frac{1}{2\pi}$. Consider the following nonlinear mapping:

$$\begin{aligned}[y_1, y_2] &= \mathbf{G}(s_1, s_2) \\ &= [s_1 \cos(s_2), s_1 \sin(s_2)]\end{aligned} \tag{2.14}$$

The joint pdf of y_1 and y_2 is then

$$
\begin{aligned}
f_{y_1,y_2}(y_1, y_2) &= \frac{1}{2\pi} e^{-\frac{1}{2}(y_1^2+y_2^2)} \\
&= \left(\frac{1}{2\pi} e^{-\frac{y_1^2}{2}}\right)\left(\frac{1}{2\pi} e^{-\frac{y_2^2}{2}}\right)
\end{aligned}
\tag{2.15}
$$

Thus, y_1 and y_2 are statistical independent despite the fact that each is a nonlinear function of both of the original source signals s_1 and s_2.

The research presented in this dissertation is most closely aligned with the nonlinear blind source separation problem because our objective is to identify and visualize nonlinear variation patterns acting on the observed data. An autoencoder containing a bottleneck layer can be interpreted as learning both an encoding function $\mathbf{G}$ mapping the observed vector $\mathbf{s}$ to a set of latent variables $\mathbf{y}$, defined by the equation $\mathbf{y} = \mathbf{G}(\mathbf{s})$, and also a decoding function $\mathbf{H}$ which maps $\mathbf{y}$ back to the high-dimensional feature space of the input as a reconstructed vector $\mathbf{s}' = \mathbf{H}(\mathbf{y})$ where $\mathbf{s}'$ denotes the matrix transpose of $\mathbf{s}$. The difficulty of the nonlinear blind source separation problem makes learning the true source signals $\mathbf{y}$ and the appropriate functional mappings a significant challenge, and the nonuniqueness of nonlinear ICA solutions illustrates how we cannot rely on measures such as the statistical independence or the correlation between the learned latent variables to identify the correct solution.

Chapter 3

IDENTIFYING NONLINEAR VARIATION PATTERNS WITH DEEP AUTOENCODERS

3.1 Introduction

With stricter customer demands, manufacturing quality assurance remains one of the most critical challenges facing US industries. To address this challenge, most major companies invest heavily in advanced measurement and data collection technology. One of the most promising emerging measurement technologies that is broadly applicable in discrete parts manufacturing is noncontact, dimensional coordinate metrology using laser and/or vision systems. Laser measurement can be either point scan (a fine laser beam is projected onto the part as a point and scanned across the part) or line scan (a laser beam is fanned into a plane, projected onto the part as a stripe, and scanned across the part). A single point scan produces a profile, whereas a series of point scans or a line scan produces point cloud data in 3D space. Both represent the surface geometry of the scanned part. Vision systems take what can be high-magnification images of the part surface, from which parametric (e.g., circles, lines) or nonparametric (e.g., edges, contours) geometric features or general visual characteristics of the part can be extracted with standard image processing algorithms. We refer to laser and/or vision systems generically as optical coordinate measuring machines (OCMM).

OCMM accuracy and throughput are now at levels that render it suitable for quality control of parts with moderately high precision. Buried in the rich and complex structure of the spatially dense OCMM data is a wealth of information on the

dimensional integrity of individual parts and on the precise nature of part-to-part variation. Identifying root causes of part-to-part variation is a fundamental objective of the six-sigma programs that are ubiquitous in industry. Although there is a large body of existing work on analyzing OCMM data, the vast majority pertains to fitting parametric geometric features to the data for individual parts (e.g., fitting a circle to the perimeter of a drilled hole or fitting two parallel planes that contain the data on a nominally flat surface). Their purpose is to verify the dimensional integrity of individual parts, and any subsequent analysis of part-to-part variation is limited to parametric variation in the specific fitted features. Moreover, the recent flurry of work on statistical process control with profile and image data is almost entirely focused on monitoring for changes in the profiles and provides very little diagnostic information on the nature of the variation. As a result, the full potential of OCMM technology is currently grossly under-utilized.

To address this deficiency, we propose a methodology to identify and visualize nonlinear variation patterns in OCMM profile data using deep multilayer neural networks. The network architecture is selected such that the system represents an autoencoder through which the original OCMM data is encoded to a small number of hidden nodes in the middle layer of the network. Each of these nodes represents a nonlinear variation pattern in the observed data, which is visualized by adjusting the value at the node used to reconstruct the signal. We demonstrate the effectiveness of this approach on both 2-D and 3-D simulated profile data containing 1-2 independent variation patterns. These results are compared to an alternative method of identifying variation patterns by applying linear methods after reindexing the profile data to produce linear patterns and also to a manifold learning approach.

The rest of this paper is organized as follows. First, the problem of identifying nonlinear variation sources in profile data is detailed and a brief survey of previous

work related to this problem is provided. We then describe an alternative method for identifying variation patterns by applying linear methods after reindexing profile measurements, which is used as a basis of comparison for the deep autoencoder method in the results section. The methodology for using deep autoencoders to discover sources of variation is then detailed, followed by a description of the results obtained from applying the method to several simulated data sets and a comparison to alternative methods. We conclude the paper with a summary of our findings and a discussion of future work.

3.2 Problem Description

The nonlinear profile data for which we seek to identify common sources of variation is assumed to have the functional form $= \mathbf{f}(\mathbf{v}_i) + \mathbf{w}_i$, where $\mathbf{x}_i$ is the i^{th} observed profile in a set of N instances. A given profile i contains M discrete height measurements taken along the surface of a part, which is expressed as $\mathbf{x}_i = [x_{1,i}, x_{2,i}, \ldots, x_{M,i}]^{\mathrm{T}}$. The profiles are defined by K variation patterns impacting the observed measurements, with the values of these variation patterns represented by the vector $\mathbf{v}_i = [v_{1,i}, v_{2,i}, \ldots, v_{K,i}]^{\mathrm{T}}$ for the i^{th} instance. These K variation patterns are mapped to the M-dimensional space of the observed height measurements by the vector of M common nonlinear functions $\mathbf{f}(\mathbf{v}_i) = [f_1(\mathbf{v}_i), f_2(\mathbf{v}_i), \ldots, f_M(\mathbf{v}_i)]^{\mathrm{T}}$. The i^{th} observed profile $\mathbf{x}_i$ is also corrupted by a small noise component $\mathbf{w}_i = [w_{1,i}, w_{2,i}, \ldots, w_{M,i}]^{\mathrm{T}}$. We only assume that the patterns can be represented by some nonlinear function $\mathbf{f}(\ )$; we do not have any specific knowledge on the nature of this function as we aim to learn it using only the observed profiles $\mathbf{x}_i$.

An example of this form of profile data is provided in figure 3.1. Six simulated profiles of a gasket bead are depicted with varying degrees of bead flattening, which represents the $K = 1$ common source of variation among the instances. Each profile

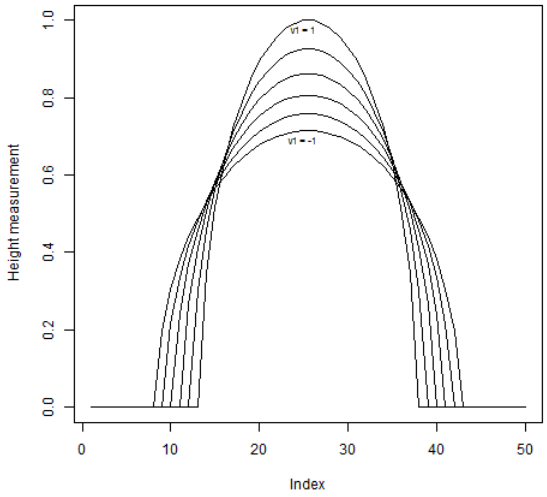

Figure 3.1: Example of Nonlinear Profile Data With $K = 1$ Variation Pattern

consists of $M = 50$ discrete height measurements taken along the surface of the gasket bead. The profiles are uncorrupted by noise in this example for illustrative purposes, allowing the i^{th} profile to be fully represented by $\mathbf{x_i} = \mathbf{f}(v_i)$. As the value of the single variation pattern v_i decreases from 1 to -1, the amount of flattening observed in the gasket bead increases and the bead becomes more elongated.

Our objective is to learn and visualize the nonlinear functions of common variation patterns $\mathbf{f}(\mathbf{v}_i)$ purely from the observed data $\mathbf{x}_i$ in order to provide an understanding of the variation sources impacting the observed profiles. For the example depicted in figure 3.1, the desired solution would be an interactive visualization tool in which a process engineer moves a slider controlling the value of v_i, resulting in reproduced profiles exhibiting various degrees of bead flattening. This allows the engineer to bypass the dimensionality of the observed height measurements (M) and instead interact solely with the lower dimensionality of common variation patterns (K), which is particularly useful when working with OCMM data where the number of observed measurements is significantly greater than the number of common variation sources.

3.3 Related Work

Principal Component Analysis (PCA) is a standard method for identifying and visualizing linear variation patterns in data (Jolliffe, 2005). Because the principal components are orthogonal, they facilitate visualization of uncorrelated variation patterns through reconstruction of the original data using a subset of the principal components evaluated over a range of values. However, PCA is limited to applications where the patterns are linear or where they can be transformed to linearity.

Kernel PCA (KPCA) can be viewed as an extension of linear PCA in which the n-dimensional data vectors (the x_i) are mapped to some M-dimensional ($M >> n$ typically, and M may be infinite) feature space via some map $\phi(x) = [\phi_1(x), \phi_2(x), , \phi_M(x)]^{\mathrm{T}}$, where each $\phi(x)$ is some scalar-valued nonlinear function of x. One then conducts PCA on the set of feature vectors $\{\phi(x_1), \phi(x_2), \ldots, \phi(x_N)\}$. In practice, the feature map is defined implicitly by some kernel function for computational reasons (Schölkopf *et al.*, 1998). Since KPCA can remove some noise in data, it can be useful for detecting underlying variation patterns and it has been applied to variation pattern analysis previously (Sahu *et al.*, 2014; Shinde *et al.*, 2014; Im *et al.*, 2012). However, it does not provide a parametric representation, nor even an explicit representation, of the patterns (only a collection of denoised data points), and hence it is not ideally suited for visualization.

Blind source separation (BSS) methods such as Independent Component Analysis (ICA) are closely related to our task of identifying unique variation patterns. Whereas PCA seeks orthogonal directions that are efficient for representing the original data, ICA searches for directions that minimize the statistical dependence between them (Comon, 1994). However, ICA discovers only linear patterns, making it unsuitable for our task of blindly identifying nonlinear variation patterns. BSS methods for the case

of linear patterns have been proposed with applications to manufacturing variability analysis (Shan and Apley, 2008; Apley and Lee, 2003).

Approaches based on manifold learning have been used for discovering variation patterns in profile data (Shi *et al.*, 2016, 2015). One disadvantage of these approaches is that new data cannot be analyzed with a previously trained model because the manifold coordinates do not provide a functional mapping of the original profiles to the low-dimensional representation. Additionally, our results show that deep autoencoders yield a better reconstruction of the original profiles from the low-dimensional representation.

Autoassociative neural networks have been proposed as a nonlinear extension of principal component analysis which can be used to perform dimensionality reduction (Kramer, 1991). The use of these models with many hidden layers for improved feature learning have been proposed recently (Hinton and Salakhutdinov, 2006). Such models are often referred to as deep autoencoders and they will be discussed in more detail in subsequent sections of the paper.

3.4 Pattern Discovery by Reindexing to Linearity

One method for learning the nonlinear variation patterns $\mathbf{f}(\mathbf{v}_i)$ in the 2D gasket bead example is the application of linear methods after the measurements have been reindexed to produce linearity, which is possible in this case but not necessarily for all patterns in general. For example, consider the gasket bead profile depicted in figure 3.2. The profile again consists of $M = 50$ discrete height measurements which have been corrupted by noise, but are influenced by the same single variation pattern depicted in figure 3.1. The red solid circles in figure 3.2 show the 11^{th} and 25^{th} indexed measurement in the profile (x_{11}, and x_{25}, respectively). Figure 3.3 shows the height measurements observed at the indexed x_{11} and x_{25} values across 200 simulated

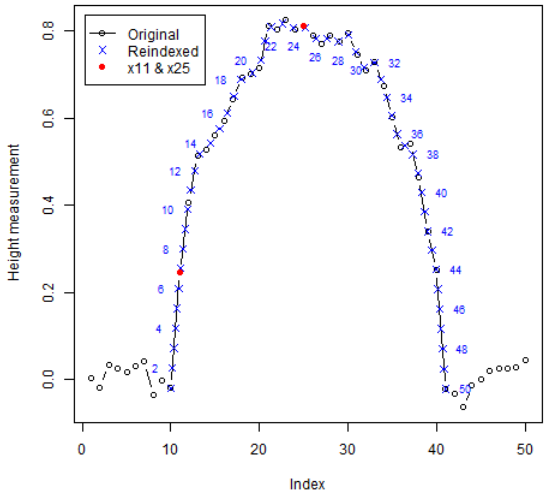

Figure 3.2: Example of Gasket Bead Profile Before and After Reindexing

profiles, which were sampled from 1 of the 6 flattening patterns depicted in figure 3.1. The relationship is highly nonlinear, indicating that linear methods would not be appropriate.

Figure 3.2 also depicts 50 reindexed height measurements across only the nonlinear portion of the gasket bead, which are represented by blue X marks on the plot. The blue numbers in the plot denote the index number of the closest reindexed point. The first reindexed point occurs where the nonlinear portion of the gasket bead begins and the last reindexed point (index 50) similarly occurs where the nonlinear portion ends. The nearest reindexed point to the original x_{11} indexed value is the 7^{th} reindexed point, while the 25^{th} reindexed point occurs at approximately the same location on the gasket bead as the original x_{25} indexed measurement. A scatter plot of the 7^{th} and 25^{th} reindexed height measurements is provided in figure 3.4, showing that reindexing has removed the nonlinearity across profiles sampled from varying degrees of bead flattening. Similar approximately linear relationships hold for other pairs of reindexed measurements, indicating that linear methods such as PCA can be applied

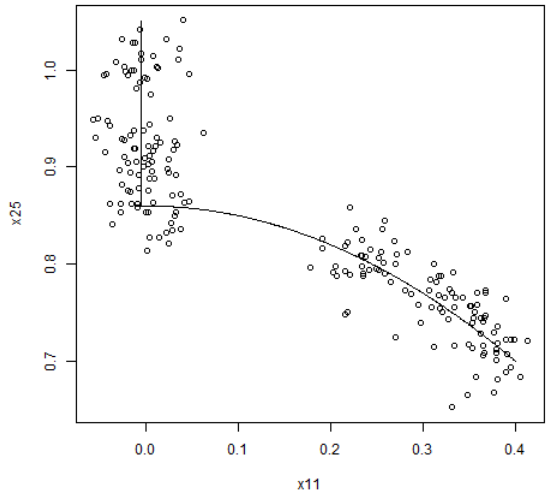

Figure 3.3: Height Measurement at the 11^{th} and 25^{th} Index Across 200 Sampled Profiles

after reindexing to discover the variation patterns.

In order to arrive at the reindexing solution depicted in figure 3.2, we use the following methodology which is based on linear interpolation. We approximate the arc length of the nonlinear portion of the gasket bead, which consists of the measurements between the change points defining the degree of the flattening variation pattern. For example, in the gasket bead profile depicted in figure 3.2, the change points occur at the indexed measurements x_{10} and x_{40}. The indices corresponding to the first and second change points are denoted c_1 and c_2. These change points are assumed to be known prior to reindexing; in practice, they could be determined using standard methods for detecting slope changes in curves such as fitting piecewise linear regression models (Howard *et al.*, 2016b).

We approximate the arc length of the gasket bead between the change points, denoted L_i for the i^{th} profile, by summing the lengths of the line segments between adjacent measurements taken along the gasket bead head. Letting $S_{j,i}$ denote the length of the line segment connecting the indexed measurement $j-1$ to measurement

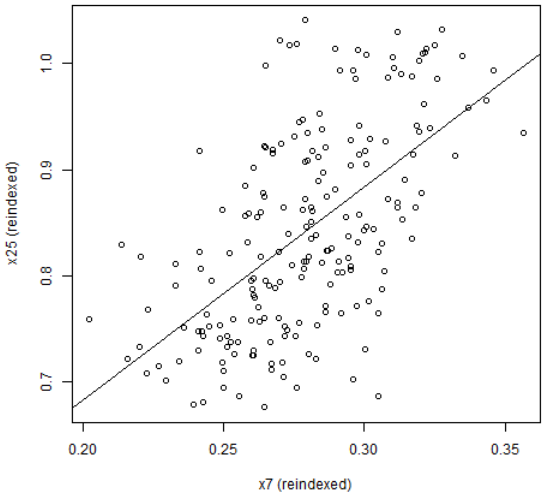

Figure 3.4: Height Measurement at the 7^{th} and 25^{th} Reindexed Points Across 200 Sampled Profiles

j on the i^{th} profile, the total approximated arc length is

$$\begin{aligned} L_i &= \sum_{j=c_1+1}^{c_2} S_{j,i} \\ &= \sum_{m=c_1+1}^{c_2} \sqrt{(x_{m,i} - x_{m-1,i})^2 + \left(\frac{m-(m-1)}{M}\right)^2} \end{aligned} \tag{3.1}$$

For completeness, we set $S_{j,i} = 0$ for $j \leq c_1$ and $j > c_2$. Note that the denominator in the second term under the square root above ensures that the horizontal displacement is calculated on the same scale as the vertical displacement for the purpose of determining curve length; this is necessary in our example because the vertical height measurements range between values of 0 and 1, whereas the horizontal index number ranges between 1 and 50. Using the approximated arc length L_i, we then find M new reindexed points along the gasket bead such that the length of the line segment connecting any adjacent pair of reindexed points is $S_i' = L_i/(M-1)$. The horizontal and vertical coordinates of the m^{th} reindexed point, denoted $x_{m,i}^H$ and $x_{m,i}^V$, are then calculated as

$$
\begin{aligned}
x^H_{m,i} &= \frac{m^* - 1}{M} + \cos(\theta)\left(S'_i(m-1) - \sum_{k=2}^{m^*-1} S_{k,i}\right) \\
x^V_{m,i} &= x_{m^*-1,i} + \sin(\theta)\left(S'_i(m-1) - \sum_{k=2}^{m^*-1} S_{k,i}\right)
\end{aligned}
\tag{3.2}
$$

where

$$
\theta = \arctan\left[M(x_{m^*,i} - x_{m^*-1,i})\right] \tag{3.3}
$$

and

$$
m^* = \min\left\{ \ell \;\middle|\; \sum_{j=2}^{\ell} S_{j,i} > S'_i(m-1) \right\} \tag{3.4}
$$

Letting $\mathbf{x}^H_i = [x^H_{1,i}, x^H_{2,i}, \ldots, x^H_{M,i}]^{\mathrm{T}}$ and $\mathbf{x}^V_i = [x^V_{1,i}, x^V_{2,i}, \ldots, x^V_{M,i}]^{\mathrm{T}}$ denote the vectors of horizontal and vertical coordinates of the reindexed version of profile i, we form the $2m$-dimensional vector $\mathbf{y}_i = [\mathbf{x}^H_i, \mathbf{x}^V_i]$ and then perform linear PCA on the collection of these N vectors. The top K principal components, where K corresponds to the number of expected variation patterns among the profiles, are then used to reconstruct the reindexed profiles. The PCA scores for the K principal components evaluated on profile i comprise the vector $\mathbf{v}_i = [v_{1,i}, v_{2,i} \ldots, v_{K,i}]^{\mathrm{T}}$, and the functions $\mathbf{f}(\ )$ correspond to the loadings of the K principal components. The reconstruction of the reindexed points can then be mapped back to the originally observed index locations by linear interpolation.

This reindexing method is a customized approach that takes advantage of the known geometry of the gasket bead profiles. Although it is useful for the purpose of benchmarking the effectiveness of other possible methods for this problem, it cannot be simply extended to different applications such as OCMM where the data is indexed over two dimensions. The next section presents a more general approach for learning nonlinear variation patterns which will be later compared to this reindexing method.

3.5 Pattern Discovery by Deep Autoencoder

Autoassociative neural networks with a single hidden layer for encoding and decoding have been used as a form of nonlinear PCA previously (Kramer, 1991), but only recently has it become feasible to train deeper neural networks with more hidden layers through pre-training techniques such as contrastive divergence (Hinton and Salakhutdinov, 2006). Deep neural networks trained as autoencoders have been primarily applied to dimensionality reduction for the purpose of data compression, particularly in applications involving images and text documents. In contrast, we propose using deep autoencoders for the purpose of extracting features that can be used to visualize nonlinear variation patterns.

Our use of multiple hidden layers differs from the standard nonlinear PCA approach of using only a single hidden layer between the input layer and the encoding layer. While multiple hidden layers may not be necessary in all cases, the ability of deep networks to extract higher-level abstract features as more layers are added motivates their use here. Additionally, our application of deep neural networks to this problem illustrates how the concept of nonlinear PCA via autoassociative neural network can be extended to deep network architectures while still maintaining the ability to interpret the effects captured at the encoded layer.

Our approach sets the number of hidden nodes in the middle encoded layer of the network equal to the number of expected variation patterns in the data (K) and then adjusts the values at these nodes to visualize the effect of the variation patterns. The weights of the autoencoder are learned in a two-step process: first, pre-training is used to find a good set of initial weights for the network (Hinton and Salakhutdinov, 2006). These weights are then fine-tuned with backpropagation. For pre-training, we use a modified form of contrastive divergence learning where small Gaussian noise is

added to sigmoid activations and the result is truncated to the range $[0, 1]$. After both stages of learning are complete, the third step of visualizing the variation patterns is conducted by treating the values of the middle nodes for a given profile as the vector $\mathbf{v_i}$ and the weights of the network as the functions $\mathbf{f}(\ )$.

The subsequent sections detail the three steps of pre-training, backpropagation, and visualization. For additional details on pre-training, the reader is directed to (Hinton and Salakhutdinov, 2006), (Hinton, 2007), and (Hinton *et al.*, 2006).

3.5.1 Step 1: Pre-Training With Contrastive Divergence

An example of the network architecture learned during the pre-training phase is provided in figure 3.5. In this example, three hidden layers with n_1, n_2, and n_3 hidden nodes are fully connected to nodes in the layers above and below, but are not connected to nodes within the same layer. The final encoded layer contains one node for each of the K expected variation patterns in the data ($K = 2$ in this example). During pre-training, sigmoid activation functions at each of the nodes in the network are used to make a stochastic decision about the state of the hidden node. Each layer of in the network and its inputs from the layer below can be viewed as a Restricted Boltzmann Machine (RBM), which allows us to greedily pre-train the weights one layer at a time using contrastive divergence (Hinton *et al.*, 2006).

We assume that the input data is normalized to fall within the range $[0, 1]$, which makes the sigmoid function an appropriate choice for the activation function of the hidden nodes. Often the sigmoid activation at a particular hidden node is used as the probability of a Bernoulli random variable that stochastically determines the state of the node during pre-training. For modeling images, this has the intuitive interpretation of having the activation function determine the probability that a pixel in the image is either *on* or *off*. Our application to normalized data generally does

not have such an interpretation at the extremes of observed values in the data, and we found empirically that adding small Gaussian noise to the sigmoid activation and truncating the output to be in the range $[0, 1]$ yields better performance. Letting $r_j \sim \mathrm{N}(0, \sigma)$ denote a mean-zero Gaussian-distributed random variable, the state of the j^{th} node in the first hidden layer during pre-training is

$$h_j = \begin{cases} 1, & \text{if } q_j + r_j > 1 \\ q_j + r_j, & \text{if } 0 < q_j + r_j < 1 \\ 0, & \text{if } q_j + r_j < 0 \end{cases} \tag{3.5}$$

where

$$q_j = \frac{1}{1 + \exp\{-a_j\}} \tag{3.6}$$

and a_j represents the net input to hidden unit j. Note that q_j is the normal activation for a sigmoid hidden unit and this definition of h_j represents a sigmoid activation corrupted by small Gaussian noise and truncated to fall within the range $[0, 1]$. To simplify notation, we now let the unsubscripted vector $\mathbf{x}$ denote a single arbitrary input instance to the network with elements x_m for $m \in \{1, \ldots, M\}$, which are individually referred to as input nodes. Letting w_{mj} denote the weight connecting the m^{th} input node x_m and the j^{th} node of the first hidden layer, the net activation is

$$a_j = \sum_{m=1}^{M} w_{mj} x_m \tag{3.7}$$

Contrastive divergence updates the w_{mj} weights by comparing the similarity between pairs of input node and hidden node states when the model is given a training instance to the same similarity measure when the model is generating data from its learned connections. Specifically, a single iteration of contrastive divergence first stochastically determines the states of the nodes in the first hidden layer using equation 3.5. A given input node x_m is then reconstructed by using the dot product of

the vector of hidden node states and the transpose of the weight vector as the net input to a sigmoid activation function representing the input unit. The reconstructed input node values x_m for $m \in \{1, \ldots, M\}$ are used to calculate new states at the hidden nodes again using the expression in equation 3.5. Letting $< x_m, h_j >^0$ denote the product of the m^{th} input node and j^{th} hidden node states during the first step of this process (i.e. when x_m is the m^{th} element of an actual training instance and not a reconstruction) and $< x_m, h_j >^n$ denote the same product after n contrastive divergence steps, the change Δw_{mj} to the weight w_{mj} is calculated as

$$\Delta w_{mj} = \gamma(< x_m, h_j >^0 - < x_m, h_j >^n) \tag{3.8}$$

where γ is a user-defined learning rate and the similarity measure $< x_m, h_j >$ is the product of states of input node x_m and hidden node h_j. Often only $n = 1$ contrastive divergence steps are used, although performing more steps can result in better pre-training. We used $n = 5$ steps in our experiments. The update rule can also be modified to incorporate weight decay and momentum analogously to their standard backpropagation implementations, both of which we used in obtaining our results. Momentum adds to the current weight vector a portion of the previous weight update in addition to the weight update calculated in the current step. Weight decay penalizes the weight update calculated at each step by the product of the weight size and a user-specified parameter. Letting α denote the momentum parameter and β denote the weight decay parameter, the weight update at epoch t is

$$\Delta w_{ij}(t) = \gamma(< x_m, h_j >^0 - < x_m, h_j >^n - \beta w_{ij}) \tag{3.9}$$

and the new weight value at step t is

$$w_{ij}(t) = w_{ij}(t) + (1 - \alpha)\Delta w_{ij}(t) + \alpha \Delta w_{ij}(t - 1) \tag{3.10}$$

Each layer in the network is pre-trained sequentially in this manner starting with the first hidden layer. Once the first layer has been pre-trained, the states for each of

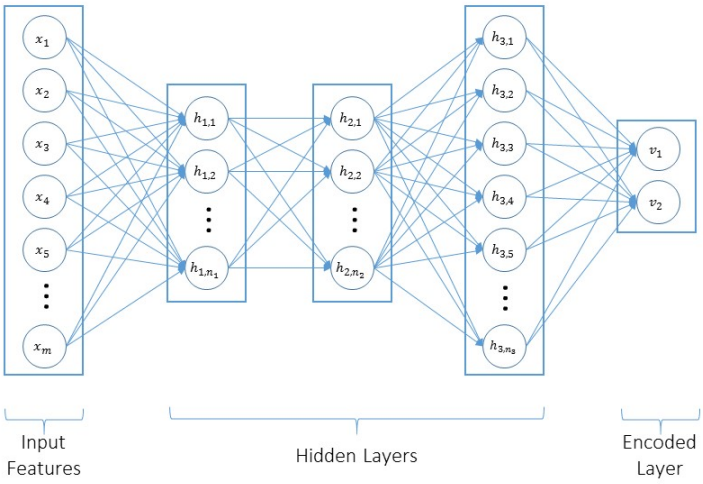

Figure 3.5: Example of Network Architecture Learned During Pre-Training

the nodes in the first hidden layer are then fixed to their normal sigmoid activation values (q_j as defined previously for an arbitrary node j) and these values are then used as the inputs for the next layer in the network. This process is continued until all connections in the network have been pre-trained.

3.5.2 Step 2: Fine-Tuning With Backpropagation

After pre-training, the learned weights are fine-tuned by backpropagating the reconstruction errors from the training data. This is done by first forming the full autoencoder network using the transpose of the weight vectors learned during pre-training as symmetric connections to an output layer of the network.

Let $\mathbf{w}^{(n)}$ denote the pre-trained vector of weights connecting nodes in the n^{th} hidden layer to the layer below. In our pre-trained network with 4 hidden layers depicted in figure 3.5, the last hidden layer has much fewer units than the M-dimensional feature space of the original data. This layer is referred to as the encoded layer be-

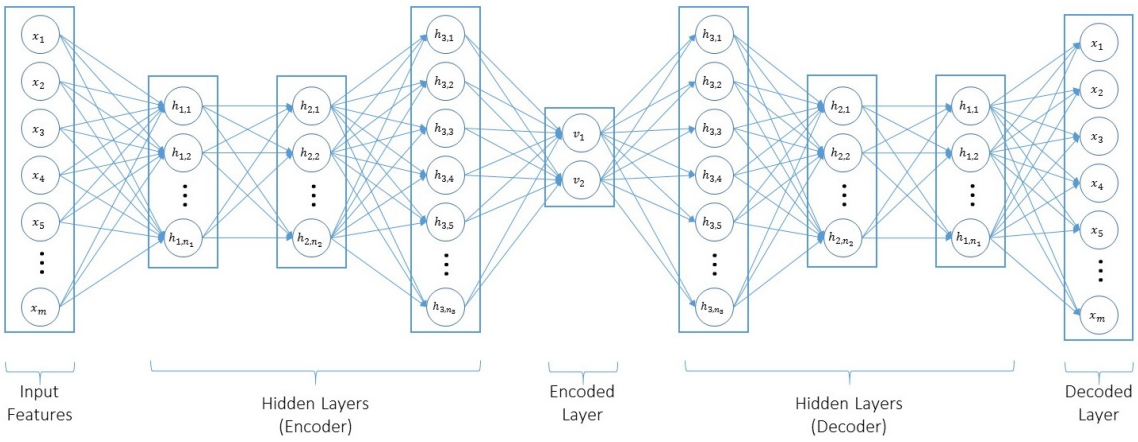

Figure 3.6: Example of Full Autoencoder Trained During Backpropagation

cause the hidden units represent an encoding of the original input vector into a low-dimensional space. To reconstruct the original input vector from the low-dimensional representation at the encoded layer, the weights are symmetrically transposed around the encoded layer to form the full autoencoder network depicted in figure 3.6. Specifically, in our example of a pre-trained network with 4 hidden layers, the weights for hidden layer n of the decoder portion of the network, $n \in \{5, \ldots, 8\}$, are initiated as $\mathbf{w}^{(n)} = \left(\mathbf{w}^{(9-n)}\right)^T$ where the superscript T denotes the matrix transpose operation. Although the autoencoder initially has symmetric weight connections around its middle encoded layer, the weights will eventually differ between the encoder and decoder portions of the network as they are fined-tuned separately using backpropagation.

We use the cross entropy error function for performing backpropagation as in (Hinton and Salakhutdinov, 2006), which performed well in our experiments and is appropriate for cases in which the output values of the network fall between 0 and 1. Letting $\hat{x}_{m,i}$ denote the m^{th} measurement produced at the decoded layer of the network for the i^{th} profile instance and $x_{m,i}$ be the actual target value for this measurement, the cross entropy error for the current set of network weights $\mathbf{W}$ is

defined as

$$J(\mathbf{W}) = \sum_{i=1}^{N} \sum_{m=1}^{M} [x_{m,i} \ln(\hat{x}_{m,i}) + (1 - x_{m,i}) \ln(1 - \hat{x}_{m,i})] \tag{3.11}$$

A portion of the training data is withheld during the fine-tuning phase to serve as a validation set for determining when to stop backpropagation. Generally backpropagation is allowed to continue until the error measured on the validation set begins to increase. The portion of the training data set used for backpropagation is divided into batches, and the weights are updated using the backpropagated errors across all training samples in a batch.

3.5.3 Step 3: Variation Pattern Visualization

To visualize the fully-trained autoencoder depicted in figure 3.6, we interpret the values evaluated at the middle encoded layer for a given profile i as the vector $\mathbf{v}_i$. Thus, $\mathbf{v}_i = [v_{1,i}, v_{2,i}]^T$ for the network with $K = 2$ encoded units that is depicted in figure 3.6. The weights connecting the middle encoded layer to the decoder hidden layers and the output layer comprise the m-dimensional function vector $\mathbf{f}(\ )$. Letting $\hat{\mathbf{x}}_i$ denote the vector of M measurements produced at the decoded layer for profile i, the visualized profile is defined as $\hat{\mathbf{x}}_i = \mathbf{f}(\mathbf{v}_i)$.

To visualize the k^{th} variation source represented by the encoded unit values $v_{k,i}$ for $i \in \{1, \ldots, N\}$, we hold the value at the other encoded layer nodes constant while varying the value of $v_{k,i}$ over its observed range in the data instances. The reconstructed profile $\hat{\mathbf{x}}_i$ produced by the varying $v_{k,i}$ values is plotted for each value to animate the effect of that variation pattern on the profile measurements. This process is repeated for each of the K variation patterns captured by the encoded layer of the network, with each pattern visualized while holding the values at the other encoded units constant.

3.6 Results

We conducted our experiments using simulated 2D and 3D profile data of a gasket bead. Two different datasets were used for the 2D case: one containing a single variation pattern characterized by bead flattening, and another containing both bead flattening and a translation pattern. A single dataset with bead flattening was used for the 3D example. All datasets also had mean-zero Gaussian noise with a standard deviation of 0.025 added to each indexed measurement of the simulated profile patterns which originally ranged in value between 0 and 1 prior to adding noise. This noise was also autocorrelated across the indexed measurements of a given profile in the 2D data sets using an autoregressive function. The autoregressive function used to impose a small amount of correlation among the noise components had the functional form $w_m = 0.25w_{m-1} + \mathrm{N}(0, 0.025)$ for $m \in \{2, \ldots, M\}$ where $\mathrm{N}(0, 0.025)$ denotes a normally distributed random variable with mean zero and standard deviation of 0.025.

We compare deep autoencoders to PCA with and without reindexing in these experiments. Additionally, comparisons to an alternative approach using manifold learning were conducted using several simulated instances of the 2D, double variation pattern gasket bead data.

In our results, we compute the mean squared error (MSE) of reconstructing the true noiseless elongation pattern using only the values at the encoded layer. The noiseless pattern would not be known in practice and is only used here to evaluate the performance of alternative methods. Letting $\hat{x}_{m,i}$ and $y_{m,i}$ denote (respectively) the reconstructed and the actual noiseless value of the m^{th} element of input vector i, $m \in \{1 \ldots M\}$ and $i \in \{1 \ldots N\}$, the MSE is defined as follows:

$$\mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} \sum_{m=1}^{M} (y_{m,i} - \hat{x}_{m,i})^2 \tag{3.12}$$

A low MSE indicates that the model is ignoring the noise in the data and capturing

the true underlying variation patterns. We compare the MSE obtained from fitting models on several different datasets using autoencoders and alternative methods in the subsequent sections.

3.6.1 2D Single Pattern Results

The 2D single pattern training dataset contained $N = 12,000$ simulated gasket bead profiles, each one represented by $M = 50$ discrete measurements taken along the profile. Although a large number of simulated profiles were used in this experiment, we illustrate in later sections how the methodology also performs well when less profiles are available. A single variation pattern of bead elongation was simulated by randomly selecting one of six distinct elongation patterns for a given profile, which are depicted in figure 3.1. Gaussian noise with autocorrelation along the profile measurements was then added to the selected elongation pattern. Thus, each of the 12,000 instances is formed from one of six elongation patterns, but each instance has its own random noise added to its pattern.

We trained an autoencoder on this data with 25 units in each of the first two hidden layers, 150 units in the third hidden layer, and 1 unit in the encoded layer. The full model including the 50-dimensional input units can be described as a 50-25-25-150-1-150-25-25-50 autoencoder, which is the same number of layers as the autoencoders proposed in (Hinton and Salakhutdinov, 2006). A single unit was used in the encoded layer to characterize the single simulated variation pattern, and the number of nodes in the other hidden layers were identified in preliminary experiments to minimize the MSE on a validation dataset. Strategies such as weight decay and monitoring for overfitting using a small amount of withheld training instances made the model less sensitive to over-specifying the number nodes in these other hidden layers of the network. While the probability of converging to a poor local minima

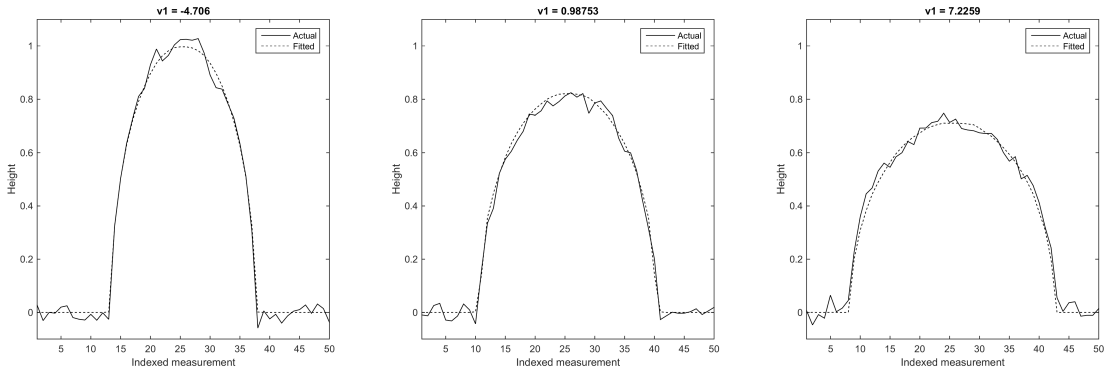

Figure 3.7: Actual and Reconstructed Profiles of 2D Single Variation Pattern Data From a 50-25-25-150-1-150-25-25-50 Autoencoder for Increasing v_1 Encoded Unit Values (Left to Right)

increases for standard neural networks as more layers are added to the network, pre-trained networks have been shown to be relatively robust to increasing network depth (Erhan *et al.*, 2009b). It is therefore simpler to err on the side of over-specifying the complexity of the model while relying on standard strategies to avoid overfitting.

Examples of actual profiles and their corresponding reconstructions from the model are provided in figure 3.7. The same models compared to the original noiseless profiles is provided in figure 3.8. Both figures show that the autoencoder does a good job of ignoring the noise while capturing the true noiseless profile pattern. The value of the encoded unit v_1, which is provided above each of the plots in figures 3.7 and 3.8, increases with the magnitude of elongation exhibited by the gasket bead. This provides an intuitive method of visualizing the variation pattern by altering the value of v_1 over its range and observing the resulting effect on the gasket bead profile.

A comparison of the test data MSE obtained by the autoencoder and two other methods is provided in table 3.1. The MSE for PCA with reindexing was obtained by interpolating back to the original profile indices from the reindexed solution so that the methods are compared on the same basis. Although we omit plots of the reconstructed profiles obtained from PCA with and without reindexing, table 3.1 clearly shows that the autoencoder outperforms both of these methods with respect

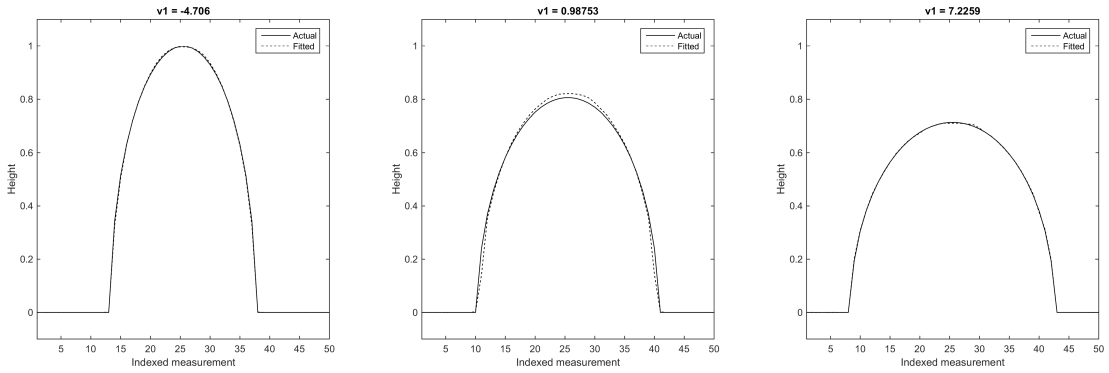

Figure 3.8: Original Noiseless and Reconstructed Profiles of 2D Single Variation Pattern Data From a 50-25-25-150-1-150-25-25-50 Autoencoder for Increasing v_1 Encoded Unit Values (Left to Right)

Method	MSE
Autoencoder	0.002012
PCA with reindexing	0.003087
PCA without reindexing	0.047566

Table 3.1: Comparison of MSE for 2D Single Variation Pattern Data Using a 50-25-25-150-1-150-25-25-50 Autoencoder, the First Principal Component of a PCA Solution With Reindexing, and the First Principal Component of a PCA Solution Without Reindexing

to the accuracy of the reconstruction. The autoencoder's MSE of 0.002012 is 35% less than that of PCA with reindexing and 96% less than regular PCA without reindexing.

3.6.2 2D Double Pattern Results

Similar to the 2D single pattern dataset, we simulated $N = 12,000$ instances of gasket bead profiles with each instance containing $M = 50$ discrete height measurements along the profile. Each instance was created by randomly selecting from among six possible elongation patterns and then shifting this profile pattern to the left or right by a random amount. Gaussian noise with autocorrelation was then added to each profile independent of the noise added to other instances in the dataset.

We trained a more complex autoencoder on this dataset with 25 units in the first

hidden layer, 75 units in the second hidden layer, 600 units in the third hidden layer, and 2 units in the encoded layer. Thus, the full model is a 50-25-75-600-2-600-75-25-50 autoencoder. A greater number of hidden layer nodes were used in this model relative to the 2D single pattern autoencoder due to the increased complexity of identifying two variation patterns. We use only 2 units in the encoded layer in order to identify a solution where the encoded units represent the two sources of variation simulated in the data, and the number of nodes in the other hidden layers were identified in preliminary experiments to minimize the MSE on a validation dataset.

In practice, the number of variation sources will be unknown, but one possible approach to selecting an appropriate number of encoded units would be to train multiple networks sequentially with each subsequent network adding 1 additional node to the encoded layer. A lack of significant reduction in the SSE of reconstruction when additional units are added could be used as a criterion for determining when the appropriate number of nodes in the encoded layer has been identified. An illustration of this approach applied to the 2D simulated gasket data with two variation patterns is provided in figure 3.9. The plot shows the MSE of reconstruction on a withheld test set for deep autoencoders trained with 1-10 nodes in the encoded layer of the network. The test MSE clearly does not improve significantly when more than 2 encoded units are used, which is expected for this example with $K = 2$ simulated sources of variation.

Figures 3.10 and 3.11 show samples of actual profiles and their reconstruction from the 2 encoded units in the model. As with the single variation pattern case, the fitted model tends to ignore the noise in the training data while capturing the true underlying pattern. More importantly, the two encoded units v_1 and v_2 can each be interpreted as characterizing one of the two variation patterns.

For example, the three samples shown in figure 3.10 all have approximately the

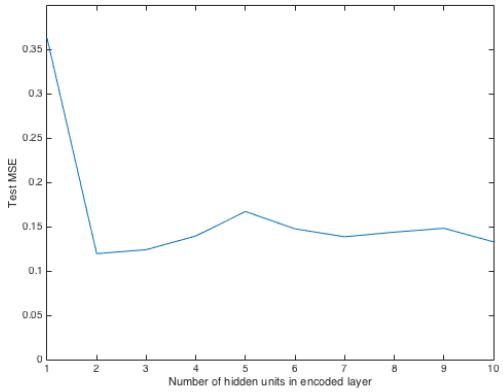

Figure 3.9: Example of the Change in Reconstruction MSE on Withheld Test Data as More Hidden Units Are Added to the Encoded Layer of a 50-25-75-600-2-600-75-25-50 Autoencoder

same value for v_2, but have increasing v_1 values from left to right in the figure. These increasing v_1 values correspond to the variation pattern which shifts the gasket bead from left to right. Similarly, the samples depicted in figure 3.11 all have approximately the same v_1 value, but have increasing values of v_2 from left to right in the figure. The elongation of the gasket bead decreases as v_2 increases, demonstrating how v_2 captures the other simulated variation pattern. Figures 3.10 and 3.11 demonstrate how the encoded units v_1 and v_2 can be used to visualize the variation patterns underlying the observed measurements along the gasket bead profile. Changing the value of one of these encoded units over its range while holding the other constant illustrates one of the two independent variation patterns used to simulate the data.

A summary of the test data MSE obtained by the autoencoder and the PCA methods on this dataset is provided in table 3.2. The autoencoder provides a better reconstruction of the noiseless profile pattern than both PCA with and without reindexing. Its MSE is approximately 25% lower than PCA with reindexing and 89% lower than regular PCA without reindexing.

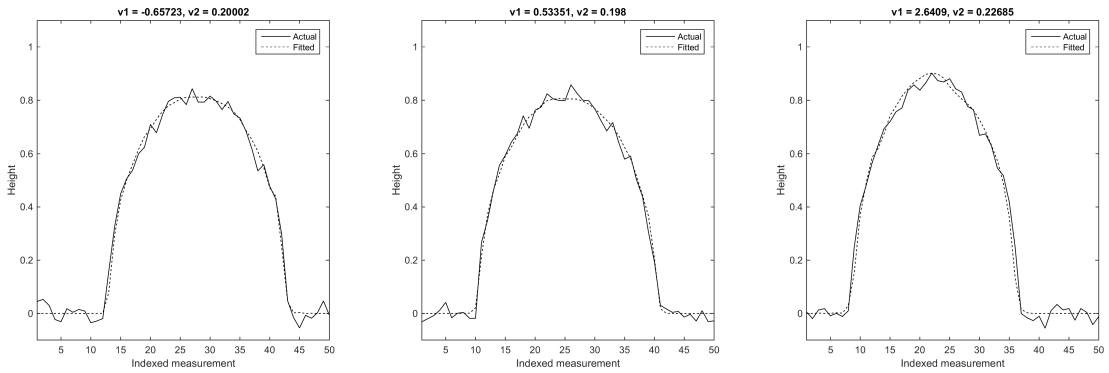

Figure 3.10: Actual and Reconstructed Profiles of 2D Double Variation Pattern Data From a 50-25-75-600-2-600-75-25-50 Autoencoder for Constant v_2 and Increasing v_1 Encoded Unit Values (Left to Right)

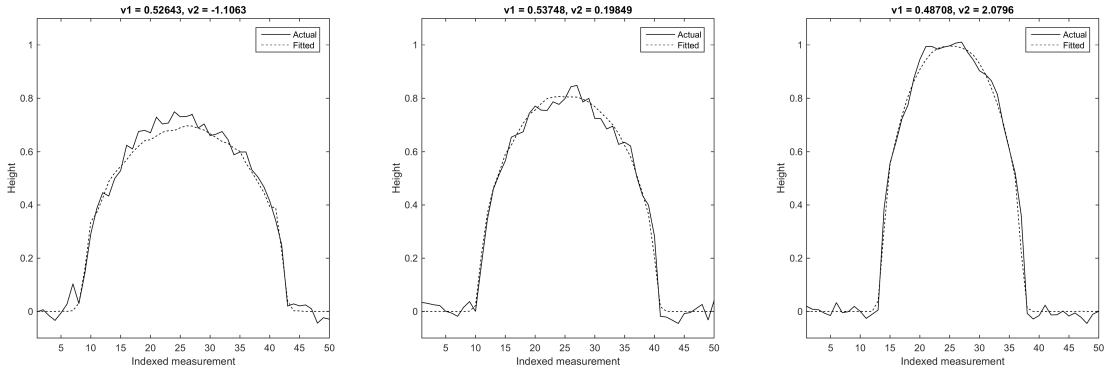

Figure 3.11: Actual and Reconstructed Profiles of 2D Double Variation Pattern Data From a 50-25-75-600-2-600-75-25-50 Autoencoder for Constant v_1 and Increasing v_2 Encoded Unit Values (Left to Right)

Method	**MSE**
Autoencoder	0.013034
PCA with reindexing	0.017472
PCA without reindexing	0.118022

Table 3.2: Comparison of MSE for 2D Single Variation Pattern Data Using a 50-25-75-600-2-600-75-25-50 Autoencoder, the Top Two Principal Components of a PCA Solution With Reindexing, and the Top Two Principal Components of a PCA Solution Without Reindexing

3.6.3 3D Single Pattern Results

The 2D gasket bead example was extended to its 3D analogue in order to test the effectiveness of the autoencoder when the dimensionality of the feature space is larger as in OCMM data. We again simulated a dataset with $N = 12,000$ instances, but each instance contains 50 the height measurements indexed along both the x- and y-axes for a total of $M = 2500$ measurements in three dimensions. A single variation pattern which represents flattening of the 3D object was simulated with additive Gaussian noise. Each instance was created by randomly selecting from one of six distinct flattening patterns and then adding the noise to the profile pattern.

We trained an autoencoder on this dataset with the same number of hidden units as was used for the 2D single pattern dataset. However, because there are 2500 input features for the 3D case, the full model is a 2500-25-25-150-1-150-25-25-2500 autoencoder.

A sample of three instances from the 3D single pattern simulated dataset is provided in figure 3.12. The profiles reconstructed for these instances by the single encoded unit v_1 from our fitted model is provided in figure 3.13. As with the 2D cases, the autoencoder ignores the noise while accurately capturing the underlying variation patterns used to generate the data. The v_1 values provided above each plot can also be used to visualize the variation pattern; increasing values of v_1 correspond to more flattening of the 3D object.

A comparison of the test data MSE produced by the autoencoder and regular PCA is provided in table 3.3. The autoencoder again outperforms PCA with respect to MSE, and the magnitude of this difference is much larger than for the 2D single pattern example. PCA with reindexing was not evaluated on this dataset because our method for reindexing cannot be easily extended to three dimensions. This highlights

Method	MSE
Autoencoder	0.00654
PCA	1.31961

Table 3.3: Comparison of MSE for 3D Single Variation Pattern Data Using a 2500-25-25-150-1-150-25-25-2500 Autoencoder and the First Principal Component of a Regular PCA Solution

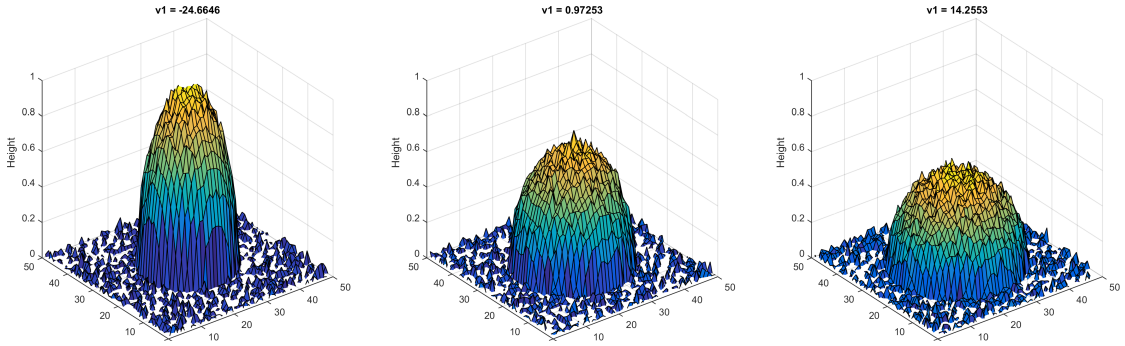

Figure 3.12: Actual Profiles of 3D Single Pattern Dataset Samples Used to Train a 2500-25-25-150-1-150-25-25-2500 Autoencoder for Increasing v_1 Encoded Unit Values (Left to Right)

an advantage of the autoencoder over other methods that require application-specific reindexing.

3.6.4 Comparison to Manifold Learning

We compared the performance of deep autoencoders to a previously-referenced manifold learning approach (Shi *et al.*, 2016) using simulated 2D profile data of a gasket bead. The profiles were simulated with two variation patterns consisting of bead flattening and a translation pattern. In order to compare the two methods across a variety of different scenarios, we simulated four types of data sets with these variation patterns. The data set types varied according to the number of profiles ($N = 200$ or $N = 1000$) and the manner in which the variation patterns were simulated. Two of the data set types had six flattening patterns and six elongation patterns from which a profile could be generated with each instance having their patterns randomly

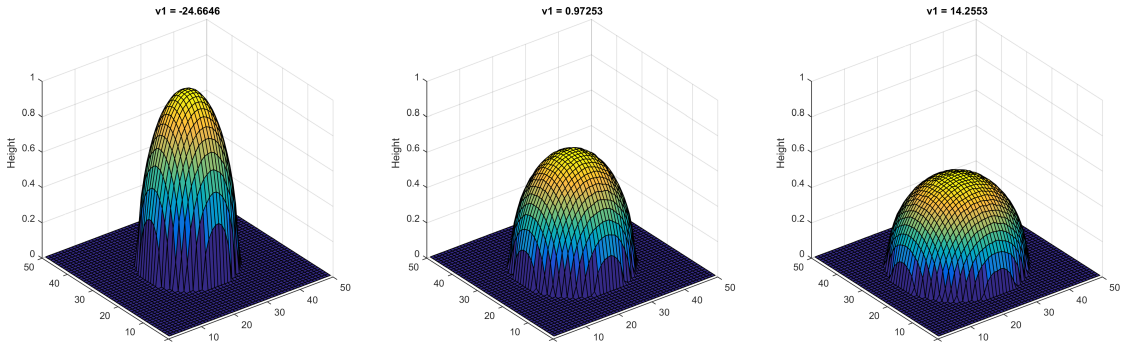

Figure 3.13: Reconstructed Profiles of 3D Single Pattern Dataset Samples From a 2500-25-25-150-1-150-25-25-2500 Autoencoder for Increasing v_1 Encoded Unit Values (Left to Right)

drawn from these discrete distributions, which we refer to as replicated instances. This could correspond to variability observed in one type of manufacturing scenario. The other two data set types had a unique elongation and flattening pattern for each instance in the data set, where the patterns for a given instance were randomly drawn from a continuous uniform distribution. This was to consider another type of manufacturing scenario. All data sets included Gaussian noise with mean zero and a standard deviation of 0.03 added to each indexed measurement of a profile, which was autocorrelated across the indexed dimension using an autoregressive process (order 1 with an autocorrelation coefficient of 0.25) so that no pair of profiles were the same. Each profile consisted of $M = 50$ discrete height measurements.

For each data set type, 10 instances of the data set were generated with each instance simulated using a different random seed. We then trained a deep autoencoder model and a manifold learning model on each instance of the data set. The deep autoencoder used the same network structure depicted in Figure 3.6 with 2 nodes in the encoded layer; the number of nodes in each hidden layer was tuned differently to each data set type. In each experiment, the entire data set was used to train the models and the noiseless versions of each profile (which were available as a byproduct of simulating the data) were used to evaluate the models' ability to learn the true un-

Data set type	ML (mean)	ML (SD)	DA (mean)	DA (SD)
Unique, $N = 200$	0.0140	0.0001	0.0130	0.0007
Replicated, $N = 200$	0.0164	0.0002	0.0123	0.0006
Unique, $N = 1000$	0.0079	0.0001	0.0070	0.0005
Replicated, $N = 1000$	0.0082	0.0001	0.0049	0.00036

Table 3.4: Comparison of Manifold Learning (ML) and Deep Autoencoder (DA) MSE Across Four Data Set Types

derlying variation pattern. Separate testing data was not withheld because a trained manifold learning model cannot be evaluated on new data.

A comparison of the mean and standard deviation (SD) of the MSE across the 10 instances in each data set type are provided in Table 3.4. The results show that the deep autoencoder yielded lower reconstruction error on average across all four data set types. The performance difference is particularly great on the data sets containing replicated variation patterns, with the deep autoencoder yielding approximately 40% lower SSE than manifold learning on the 1000 instance data set and 25% lower SSE on the 200 instance data set.

3.7 Conclusion and Future Work

We presented a deep autoencoder approach to identify nonlinear variation patterns in manufacturing data. Results from simulated 2D datasets as well as a 3D dataset show that our approach provides a very close reconstruction of the actual profile pattern from only a small number of encoded units. With respect to the MSE of reconstruction, our method consistently outperforms PCA-based methods as well as a manifold learning approach for discovering variation patterns. In addition to providing a more efficient low-dimensional representation, deep autoencoders do not rely on any known geometry in the problem domain which is necessary to apply linear

re-indexing methods. Unlike manifold learning approaches, deep autoencoders also provide a functional map between the low-dimensional representation and the original feature space, allowing the model to be applied to new data instances not used during training.

In our experiments, the encoded units produced by the autoencoder can each be interpreted as representing one of the distinct variation patterns simulated in the data, allowing for intuitive visualization of variation sources by independently varying the encoded values over their range. Although the deep autoencoder learned the true variation sources used to simulate the data in our experiments, the traditional objective functions used to train an autoencoder by minimizing the reconstruction error do not provide any guarantee that the model will learn distinct variation sources in the low-dimensional space. When more than two variation sources are present, it is possible for the autoencoder to learn features which each represent a combination of the true variation sources impacting the data. In future work, we would like to explore alternative training methods which encourage the learning of solutions where the encoded units represent distinct features.

Chapter 4

DISTINCT VARIATION PATTERN DISCOVERY USING ALTERNATING NONLINEAR PRINCIPAL COMPONENT ANALYSIS

4.1 Introduction

Discovering nonlinear variation patterns is an important problem in many applications involving high-dimensional data. For example, a manufacturing engineer may wish to understand the nature of part-to-part variation using profile quality control data for manufactured components. Such data often arises from ocular coordinate measuring machines, where a laser beam is fanned into a plane, projected onto a part as a stripe, and scanned across the part to produce a 3D point cloud. The number of measurements in the resulting profile is much larger than the number of variation patterns affecting the manufacturing process that produced it. Thus, reducing the dimensionality of the observed data to the number of underlying variation patterns can potentially shed light on the nature of the variation patterns themselves by analyzing the relationship between the reduced feature space and the observed data.

PCA is frequently used to extract low-dimensional features from high-dimensional data sources. Because the principal components are orthogonal, they can be used to identify uncorrelated linear variation patterns in high-dimensional data by interpreting each principal component as a source of variation. The order of the principal components further facilitates variation pattern analysis, because it conveys information on the amount of variation in the original data explained by each principal component. However, PCA is a linear method that fails to accurately characterize the underlying variation patterns when they are nonlinear in nature.

Our objective is to identify and visualize nonlinear variation sources in multivariate data. The examples presented in this paper involve spatially-arranged multivariate data, but the methodology can be applied more generally. The nonlinear multivariate data has the functional form $\mathbf{x_i} = \mathbf{f}(\mathbf{v_i}) + \mathbf{w_i}$, where $\mathbf{x_i}$ is the i^{th} observed instance in a set of N observations. In a manufacturing context, each data instance $\mathbf{x_i}$ could represent a profile of M discrete height measurements taken along the surface of a part, which is expressed in vector notation as $\mathbf{x_i} = [x_{i,1}, x_{i,2}, \ldots, x_{i,M}]^T$. Underlying each instance $\mathbf{x_i}$ are P variation patterns represented by the vector $\mathbf{v_i} = [v_{i,1}, v_{i,2}, \ldots, v_{i,P}]^T$. These P variation patterns are mapped to the M-dimensional space of the observed height measurements by the vector of M common nonlinear functions $\mathbf{f}(\mathbf{v_i}) = [f_1(\mathbf{v_i}), f_2(\mathbf{v_i}), \ldots, f_M(\mathbf{v_i})]^T : \mathbb{R}^P \mapsto \mathbb{R}^M$. The i^{th} observed instance $\mathbf{x_i}$ is also corrupted by a noise component $\mathbf{w_i} = [w_{i,1}, w_{i,2}, \ldots, w_{i,M}]^T$. Learning the underlying nonlinear variation patterns corresponds to identifying $\mathbf{v_i}$ and the functions $\mathbf{f}(\mathbf{v_i})$ using only the observed instances $\mathbf{x_i}$.

The model that we use to identify and visualize the nonlinear variation sources is an autoassociative neural network (ANN), which has been proposed as a form of nonlinear principal component analysis (NLPCA) (Kramer, 1991). These models are neural networks which aim to reconstruct their input after passing hidden node activations through a bottleneck layer with far fewer nodes than the dimensionality of the original data, thereby forcing data compression. The hidden nodes in the bottleneck layer will be referred to as bottleneck nodes, or alternatively as the (nonlinear) components of the model. An important issue with NLPCA methods is that often multiple bottleneck nodes align with the same variation pattern early in training (Kramer, 1991), resulting in a solution where each of the extracted features represents a combination of the true variation sources rather than distinct sources of variation. Existing approaches for NLPCA models fail to address the task of learning

a solution where each of the nonlinear components represents a single distinct source of variation in the data.

We present a new ANN learning methodology called alternating nonlinear principal component analysis (ANLPCA) which aims to addresses this deficiency. Unlike NLPCA, ANLPCA uses an alternating penalty term during backpropagation training to guide the model towards a solution where the extracted patterns are distinct and well-separated among the bottleneck nodes. Experimental results indicate that the nonlinear features extracted from the data using our ANLPCA methodology are more interpretable and better facilitate visualizing each of the true variation sources individually.

The remainder of this paper is organized as follows. In Section II, we review related literature on extracting distinct nonlinear feature representations of high-dimensional data. Our ANLPCA method is detailed in Section III, along with a new metric that we propose for measuring the separation of the variation sources among the learned features. Section IV presents experimental results from applying ANLPCA to simulated point cloud data and also to the MNIST handwritten digits dataset. Concluding remarks and a discussion of future work is provided in Section V.

4.2 Related Work

Kernel PCA (KPCA) is an extension of linear PCA where the data vectors $\mathbf{x_i}$ existing in M-dimensional space are mapped to some N-dimensional ($N >> M$ typically, and N may be infinite) feature space via map $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), , \phi_N(\mathbf{x})]^T$, where each $\phi(\mathbf{x})$ is some scalar-valued nonlinear function of $\mathbf{x}$. PCA is then performed on the set of feature vectors $\{\phi(\mathbf{x_1}), \phi(\mathbf{x_2}), \ldots, \phi(\mathbf{x_N})\}$. The feature map is defined implicitly in practice by some kernel function for computational efficiency (Schölkopf

et al., 1998). Because KPCA can remove some noise in data, it can be useful for detecting underlying variation patterns and it has been applied to variation pattern analysis previously (Sahu *et al.*, 2014; Shinde *et al.*, 2014; Im *et al.*, 2012). However, it does not provide a parametric representation, nor even an explicit representation, of the patterns (only a collection of denoised data points), and therefore it is not ideally suited for our task of visualizing the nature of the variation patterns.

Blind source separation (BSS) methods such as Independent Component Analysis (ICA) are related to our task of identifying distinct variation patterns. Whereas PCA seeks uncorrelated components that are efficient for representing the original data, ICA searches for a solution which minimizes the statistical dependence between the components (Comon, 1994). However, traditional ICA methods only discover linear patterns, making them unsuitable for our task of blindly identifying nonlinear variation patterns. BSS methods for the case of linear patterns have been proposed with applications to manufacturing variability analysis in (Shan and Apley, 2008) and (Apley and Lee, 2003). Nonlinear methods for BSS have received less attention; unlike the linear case, nonlinear ICA solutions are highly nonunique and differ greatly from the solution to the much harder nonlinear BSS problem because any arbitrary function of a pair of independent random variables will also be independent. (Jutten and Karhunen, 2003). Because our objective is the visualization of the true variation sources underlying the observed data, a nonlinear ICA solution is insufficient for extracting the features we seek because the original variation patterns can be mixed among the extracted features while still being statistically independent. Additionally, nonlinear BSS methods focus on extracting the original sources whereas we also want to learn the nonlinear functions $\mathbf{f}(\mathbf{v_i})$ which map the sources back to the original feature space for the purpose of visualization.

NLPCA is a nonlinear extension of PCA which uses an ANN to learn a compact

representation of high-dimensional data (Kramer, 1991). The neural network takes the original data as input and is trained to reproduce this input at its output layer, which has the same number of nodes as the dimensionality of the original data. The network typically has a total of four layers (excluding the input), which consists of three hidden layers and the output layer. An example of this network structure is provided in Figure 4.1. The second hidden layer is referred to as the bottleneck layer and has fewer hidden nodes than the dimensionality of the original data, forcing the network to learn a more compact representation of the data by encoding it to the units in the bottleneck layer. The network is trained to minimize the sum of squared errors in reconstructing the original data at the output layer. Unlike linear PCA, the training process for NLPCA does not guarantee that the nonlinear components represented by the bottleneck layer will be uncorrelated (or distinct in any manner) and the nonlinear components do not have any intrinsic ordering with respect to the amount of variation they characterize.

Several variants of NLPCA have been proposed. Circular PCA implements NLPCA where the units in the bottleneck layer are constrained to lie on the unit circle so that a pair of bottleneck nodes can be described by a single angle parameter (Kirby and Miranda, 1996). Inverse NLPCA learns only the decoding portion of the network (i.e., the bottleneck layer through the output layer) by treating the bottleneck node values themselves as model parameters to be learned during backpropagation, in addition to the weights (Scholz *et al.*, 2008). Deep autoencoders are ANNs that have more hidden layers than the standard NLPCA model, but that still have a single bottleneck layer in the middle of the network, which forces compression of the data (Hinton and Salakhutdinov, 2006). Unsupervised pre-training of the autoencoder is used prior to backpropagation to increase the speed of learning for deep network architectures. Pre-trained deep autoencoders have been successfully applied to learning

efficient low-dimensional representations of nonlinear profile data, outperforming alternative methods based on re-indexing and manifold learning (Howard *et al.*, 2015). However, none of these methods specifically consider the problem of ensuring that the learned components each represent distinct variation patterns in the original data.

Hierarchical nonlinear principal component networks (HNPCN) is another variant of NLPCA which seeks a solution in which a hierarchical ordering of the components exists, similar to the eigenvalue ordering of principal components in linear PCA (Saegusa *et al.*, 2004; Scholz and Vigário, 2002). The method proposes training multiple sub-networks sequentially, where each additional sub-network contains one more unit in the bottleneck layer than the previous networks, and all but one of the bottleneck layer nodes obtains its value from a previous sub-network. While this provides some order to the learned features, it requires the training of P networks (where P is the number of desired components) and does not allow the encoding portion of a network to be further trained once training has proceeded to the next sub-network. Additionally, the primary concern of the above referenced papers is learning a hierarchical ordering of the nonlinear components, which differs from our objective of identifying distinct and interpretable components. The authors did not provide experimental results showing the performance of their methods with respect to separating the nonlinear variation sources over multiple different random initializations of the network weights.

ANNs are ideal for our task of visualizing the underlying variation patterns because they can learn both the original variation sources and a functional mapping (i.e. the function $\mathbf{f}(\ )$ defined in Section I) of those sources back to the dimensionality of the observed data. Unlike the existing NLPCA methods based on ANNs, our ANLPCA model encourages the bottleneck layer nodes to each learn distinct variation patterns in the data while requiring the training of only a single network architecture. It also

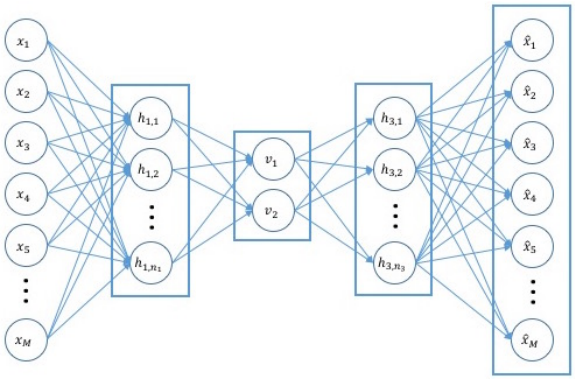

Figure 4.1: Network Architecture for ANLPCA With $P = 2$ Components

provides a hierarchical ordering of the learned components similar to PCA while allowing all model parameters to be fine-tuned during later stages of training, unlike the existing HNPCN method.

4.3 Methodology

4.3.1 Alternating NLPCA

To simplify notation, we present our methodology for ANLPCA with $P = 2$ nonlinear components in the bottleneck layer. However, the method applies directly to cases in which $P > 2$ components are desired by adding more terms to the objective function, analogous to the method presented here.

The network architecture for ANLPCA with $P = 2$ components is depicted in Figure 4.1. In this figure, $h_{k,j}$ denotes the j^{th} hidden node in layer k, and n_k denotes the number of nodes in hidden layer k. The two nodes in the bottleneck layer representing the nonlinear components are denoted by v_1 and v_2 because they are used to represent the two unknown variation patterns comprising the vector $\mathbf{v}$. The arcs represent the weights connecting nodes in one layer to the net input of nodes in the next layer; we denote $w_{k,ij}$ as the weight connecting the i^{th} node of layer $k-1$ to the j^{th} node of layer k.

To train the model on a dataset consisting of N instances, we minimize the following objective function with respect to the weights:

$$J(\mathbf{W}) = \frac{1}{2}\left[\sum_{i=1}^{N}\sum_{m=1}^{M}(x_{i,m} - \hat{x}_{i,m})^2\right] + \lambda J_r(\mathbf{W}) \tag{4.1}$$

In Equation 4.1, $x_{i,m}$ denotes the m^{th} component of instance i in the training set, $\hat{x}_{i,m}$ is the value of $x_{i,m}$ reconstructed at the output layer of the neural network, and $\mathbf{W}$ denotes the set of all weights used in the network. The $\lambda J_r(\mathbf{W})$ term in Equation 4.1 is a relaxation of a constraint on learning which alternates depending on the current epoch of training, where the training epoch is denoted by r. Letting R be a user-specified parameter, we define $J_r(\mathbf{W})$ as:

$$J_r(\mathbf{W}) = \begin{cases} \left[\sum_{j=1}^{n_3}(w_{3,1j} - \bar{w}_{3,1j})^2 \right. \\ \left. + \sum_{j=1}^{n_1}(w_{2,j1} - \bar{w}_{2,j1})^2\right], & \text{if } r \in \Omega_1 \\ \left[\sum_{j=1}^{n_3}(w_{3,2j} - \bar{w}_{3,2j})^2 \right. \\ \left. + \sum_{j=1}^{n_1}(w_{2,j2} - \bar{w}_{2,j2})^2\right], & \text{if } r \in \Omega_2 \end{cases} \tag{4.2}$$

where

$$\begin{aligned} \Omega_1 &= \{r | r \leq R \vee [r > 2R \wedge r(\text{mod}2) = 0]\} \\ \Omega_2 &= \{r | R < r \leq 2R \vee [r > 2R \wedge r(\text{mod}2) = 1]\} \end{aligned} \tag{4.3}$$

In Equation 4.3, $\bar{w}_{k,ij}$ is set equal to the value of $w_{k,ij}$ obtained from the previous iteration of backpropagation. That is, if $w_{k,ij}^{(r)}$ denotes the new value of $w_{k,ij}$ after the r^{th} backpropagation iteration, then in iteration $r+1$, we set $\bar{w}_{k,ij} = w_{k,ij}^{(r)}$. The parameter λ in Equation 4.1 is set equal to a large value which can be tuned using cross validation.

We use conjugate gradient descent to iteratively minimize the objective function in Equation 4.1. The first term in Equation 4.1 is simply the squared error loss function

that is commonly used when training ANNs and in NLPCA. The other term $J_r(\mathbf{W})$ represents a constraint imposed on the objective function by our ANLPCA method to arrive at a solution where each node in the bottleneck layer represents a distinct variation pattern, as explained below.

The objective function term $J_r(\mathbf{W})$ only affects the weights entering into and leaving the bottleneck layer, $w_{2,ij}$ and $w_{3,ij}$. Thus, the weight updates for $w_{1,ij}$ and $w_{4,ij}$ in the first and fourth layers are the same as in regular NLPCA. The first definition of $J_r(\mathbf{W})$ in Equation 4.3 for $r \in \Omega_1$ only involves the weights coming into and out of v_1 in the bottleneck layer, and the relevant partial derivatives for this term are:

$$\begin{aligned} \frac{\partial J_r(\mathbf{W})}{\partial w_{3,1j}} &= 2(w_{3,1j} - \bar{w}_{3,1j}) \\ \frac{\partial J_r(\mathbf{W})}{\partial w_{2,j1}} &= 2(w_{2,j1} - \bar{w}_{2,j1}) \end{aligned} \tag{4.4}$$

The second definition of $J_r(\mathbf{W})$ in Equation 4.3 for $r \in \Omega_2$ similarly involves only the weights going into and out of v_2. Thus, the relevant partial derivatives for the second definition are:

$$\begin{aligned} \frac{\partial J_r(\mathbf{W})}{\partial w_{3,2j}} &= 2(w_{3,2j} - \bar{w}_{3,2j}) \\ \frac{\partial J_r(\mathbf{W})}{\partial w_{2,j2}} &= 2(w_{2,j2} - \bar{w}_{2,j2}) \end{aligned} \tag{4.5}$$

The effect of $J_r(\mathbf{W})$ on the weight updates is, therefore, simply to add the terms in Equation 4.4 and 4.5 to the usual squared error loss gradients for the relevant weights entering and leaving the bottleneck layer. Alternating the definition of the $J_r(\mathbf{W})$ term based on the training epoch results in only updating the weights connected to a single bottleneck node in any given epoch of learning. The weights connected to the other bottleneck node remain approximately constant due to the active penalty on deviations from the prior iteration's weights for that node.

Allowing only the weights connected to v_1 to be updated for the first R epochs

allows the error associated with the largest variation in the data to be learned solely by v_1. After the largest variation source has been learned, the penalty alternates and encourages the v_1 weights to remain constant while v_2 learns the next largest variation source. Alternating the penalty between the two nodes after the first $2R$ epochs are complete allows the model to continue updating the weights on each node independently to fine-tune the model, which is otherwise not possible if we try to learn the weights for a given bottleneck node entirely before moving on to train the next node. To extend the model to instances in which there are $P > 2$ bottleneck nodes, we simply use an analogous alternating definitions of the $J_r(\mathbf{W})$ term for each of the $k = 1, \ldots, P$ bottleneck nodes. This would ensure that only the weights connected to a single bottleneck node are learned during an iteration of training.

4.3.2 Tangent Vector Cosine Similarity Metric

After training an ANN, the sum of squared errors (SSE) represented by the first term in Equation 4.1 is typically calculated on a test dataset in order to provide a generalized estimate of the quality of fit produced by the model. This reconstruction error is important for our task of visualizing the variation patterns because it indicates how well the actual data are modeled. However, we also need to evaluate a solution based on how distinct are the variation patterns modeled by each of the bottleneck nodes, which requires another metric in addition to the standard SSE.

To measure the degree to which the bottleneck nodes v_1 and v_2 capture distinct variation patterns in the original data, we introduce a metric called the tangent vector cosine similarity (TVCS). TVCS has similarities with the concept of tangent distance, which is a transformation-invariant distance measure used in image recognition (Simard *et al.*, 1993); however, the tangents are used for a different role here.

After training the neural network, each of the $i = 1, \ldots, N$ instances in our dataset

have distinct bottleneck node values $\mathbf{v}_i = [v_{i,1}, v_{i,2}]^T$ in addition to a reconstructed vector $\mathbf{f}(\mathbf{v}_i) = \hat{\mathbf{x}}_i$, where $\mathbf{f}(\mathbf{v}_i) : P \rightarrow M$ denotes the nonlinear functional mapping of $\mathbf{v}_i$ to the original M-dimensional space produced by the decoder portion of the network (i.e. the weights and activation functions from the bottleneck layer to the output layer). The functional mapping $\mathbf{f}(\mathbf{v}_i)$ for $i \in \{1, \ldots, N\}$ defines a P-dimensional manifold in the M-dimensional space of the original data describing the possible reconstructions that can be produced by the model.

Let $\bar{v}_1$ and $\bar{v}_2$ denote the midpoints of the range of these bottleneck node values evaluated across the N instances after training. If we fix $v_2 = \bar{v}_2$, the collection of points $\mathbf{f}([v_{i,1}, \bar{v}_2]^T)$ for $i = 1, \ldots, N$ represent a path along the manifold defined by v_1. Similarly, the points $\mathbf{f}([\bar{v}_1, v_{i,2}]^T)$ for $i = 1, \ldots, N$ represent a different path along the manifold which characterizes the effect the second bottleneck node v_2 exerts on the reconstructions. For the purpose of measuring the distinctness of the patterns represented by v_1 and v_2, we approximate these paths by their tangent vectors T_1 and T_2 to the manifold at the midpoint values of the bottleneck nodes $\bar{\mathbf{v}} = [\bar{v}_1, \bar{v}_2]^T$ using the method of finite differences.

Because the tangent vectors T_1 and T_2 represent linear approximations of the directions in the manifold defined by the bottleneck nodes v_1 and v_2, the orthogonality of these vectors indicates that the bottleneck nodes have learned variation patterns in different directions of the original M-dimensional space. This is desirable because we want the nodes to characterize different information about the variation sources in the data. It is also similar to the orthogonality of loadings used in PCA to describe different sources of variation. Thus, we measure the cosine of the angle between the tangent vectors T_1 and T_2, which is close to zero when the tangent vectors have this desirable characteristic of being nearly orthogonal and increases in absolute value towards one as they become less orthogonal. The TVCS measure is defined as:

$$TVCS = \left| \frac{T_1 \cdot T_2}{||T_1||||T_2||} \right| \tag{4.6}$$

4.3.3 Enhancements to TVCS

The orthogonality of tangent vectors on the manifold is analogous to the condition of orthogonal loadings in PCA. ANN models with a small TVCS therefore share the intuitive property of having components that characterize orthogonal directions in the original feature space, as in PCA. Unfortunately the orthogonality of the tangent vectors T_1 and T_2 is not a sufficient condition for the variation patterns to be well-separated between the bottleneck nodes v_1 and v_2. This is because the true variation patterns can be completely mixed between v_1 and v_2 while still producing patterns that are independent in the M-dimensional space of the observed data (Jutten and Karhunen, 2003), as discussed in Section II.

However, we can improve the usefulness of TVCS for measuring source separation if we have prior information about one of the nonlinear variation patterns present in the data. Specifically, if we know the direction in the original M-dimensional space primarily impacted by one of the true variation patterns, reflecting one of the tangent vectors about this direction will reveal if the variation sources have been mixed between the bottleneck nodes. We therefore calculate the following second TVCS measure.

$$TVCS_2 = \left| \frac{T_1 \cdot T_2^R}{||T_1||||T_2^R||} \right| \tag{4.7}$$

where T_2^R is the tangent vector to the manifold defined by the second bottleneck node v_2 after each of the reconstructed instances $\mathbf{f}([\bar{v}_1, v_{i,2}]^T)$ have been reflected about the direction in the M-dimensional space impacted by the known variation source. Letting $TVCS_1$ denote the original TVCS measure defined in Equation 4.6 previously, the final measure incorporating the prior information about one of the

known variation sources is defined as $TVCS = \max(TVCS_1, TVCS_2)$.

To illustrate, consider the reconstructed instances of our simulated point cloud data depicted in Figure 4.2. There are two distinct variation patterns that are simulated into the data, with additional small Gaussian noise added to the instance. Each instance is a 2500-dimensional point cloud representing a bowl-shaped object in 3-dimensional space. One of the variation patterns acting on this object is a flattening pattern, which expands the object symmetrically along the x and y axes while flattening its height in the z dimension. The second variation pattern, independent to the first, shifts the object along the horizontal axis.

Figure 4.2 shows shaded surface plots of the object viewed from above (i.e. looking down from the top of the z axis, as in a contour plot). These plots were obtained from an ANLPCA solution in which v_1 accurately characterizes the first variation pattern and v_2 characterizes the second pattern. The top three plots in Figure 4.2 illustrate how the object flattens as the value of v_1 decreases and v_2 remains constant at its midpoint. The bottom three plots illustrate the shifting of the object from left-to-right along the horizontal axis as the value of v_2 increases and v_1 remains fixed at its midpoint. The tangent vectors T_1, T_2, and T_2^R for this ANLPCA solution are plotted in Figure 4.3.

Now consider the regular NLPCA solution with random starting weights for this same dataset depicted in Figure 4.4. These plots represent a mapping of the $\mathbb{R}^{2500}$ space of the model's reconstructions to a 50×50 grid corresponding to the physical space in which the object resides (hereafter referred to as the 'visualization space'). For a given model reconstruction $\mathbf{f}(\mathbf{v})$ produced by the bottleneck node values $\mathbf{v} = [v_1, v_2]$, we denote the (i, j) coordinate pair of the reconstruction in the visualization space as $g_{\mathbf{v}}(i, j)$ for $i \in \{1, \ldots, 50\}$, $j \in \{1, \ldots, 50\}$. Note that in the NLPCA solution depicted in Figure 4.4, v_1 represents both the flattening pattern and shifting

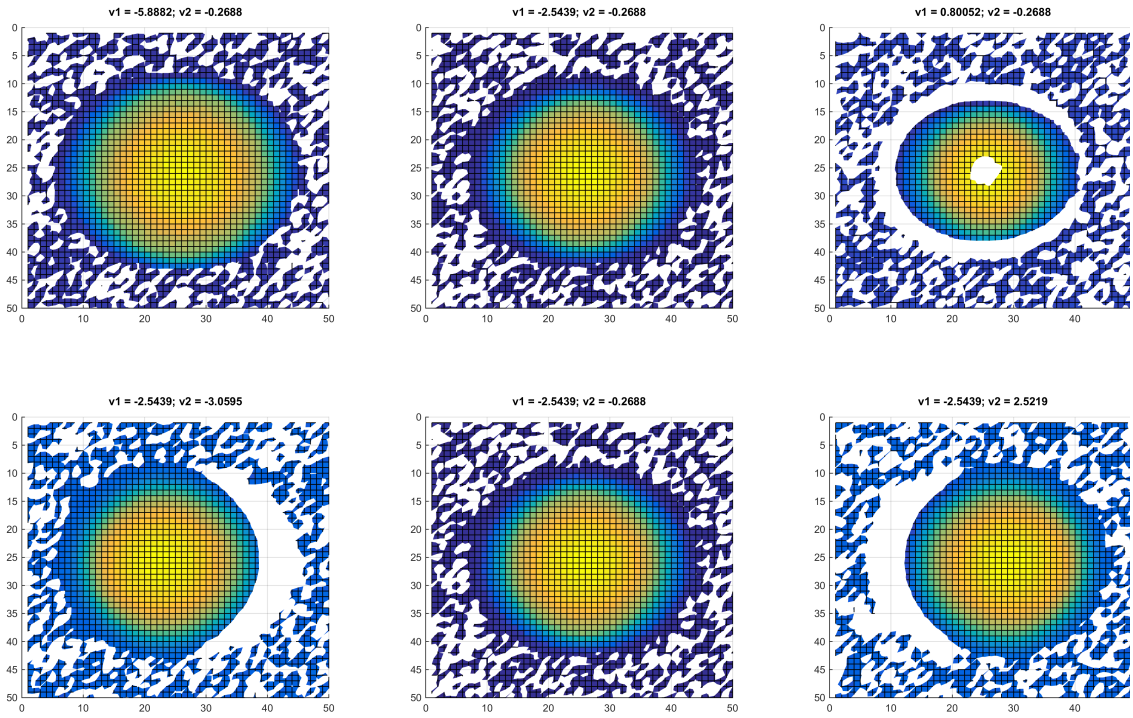

Figure 4.2: Example of ANLPCA Reconstructions of Simulated Point Cloud Data for Constant v_2, Increasing v_1 (Top, Left to Right) and Constant v_1, Increasing v_2 (Bottom, Left to Right)

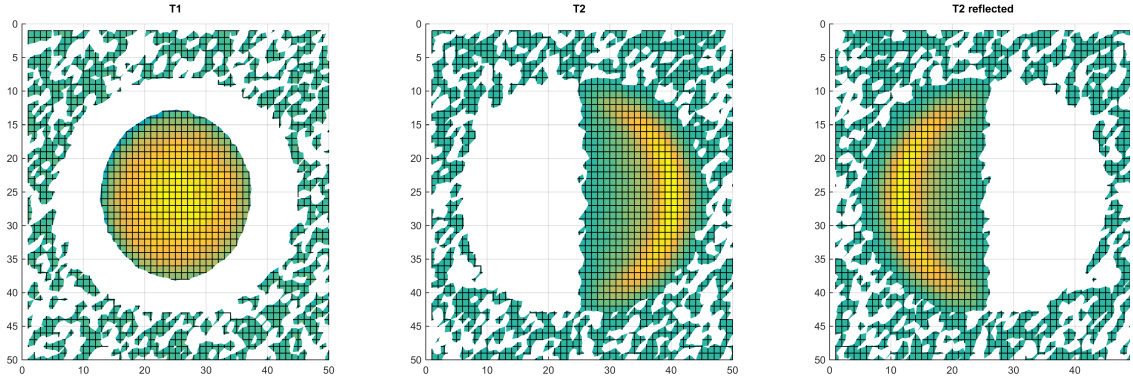

Figure 4.3: Tangent Vectors From an ANLPCA Solution Evaluated at a Single Point

the object to the right along the horizontal axis of the visualization space, as depicted by the top three plots of Figure 4.4. The bottom three plots show that v_2 has also learned a combination of these two variation patterns where the reconstructions are produced over a different region of the visualization space.

The fact that each of the bottleneck nodes has learned a combination of the two true variation patterns is revealed by how the reconstructions are identical after reflecting the patterns produced by one of the bottleneck nodes around the vertical axis in the reconstruction space. Specifically, let $\mathbf{v} = [\bar{v}_1 + \delta, \bar{v}_2]$ and $\mathbf{v}' = [\bar{v}_1, \bar{v}_2 - \delta]$ represent two sets of bottleneck node values where δ is a small deviation from one of the elements in each set. Figure 4.4 illustrates how $g_{\mathbf{v}}(i, j) \approx g_{\mathbf{v}'}(50 - i, j)$, which defines approximate equivalence after reflecting the reconstruction associated with v_2 around the vertical axis of the visualization space at the $i = 25$ point of the horizontal axis.

The tangent vectors for this NLPCA solution are depicted in Figure 4.5 and illustrate that, despite the variation patterns being mixed among v_1 and v_2, the tangent vectors T_1 and T_2 are still nearly orthogonal because they point in different directions of the 2500-dimensional feature space, resulting in a small value of $TVCS_1$. However, when the reconstructed values for one of the bottleneck nodes are reflected around the vertical axis of the visualization space, the reflected tangent vector T_2^R is approximately equal to the negative of T_1, yielding a value of $TVCS_2$ that is near 1. Thus, taking the maximum of $TVCS_1$ and $TVCS_2$ yields a large value near 1 when the bottleneck nodes model a combination of the variation patterns, and it produces a small value near 0 when they each represent distinct patterns.

This formulation of the $TVCS$ measure relies on prior knowledge that one of the variation patterns primarily affects the object in the direction of the horizontal axis of the visualization space. However, different rotations of the visualization space could

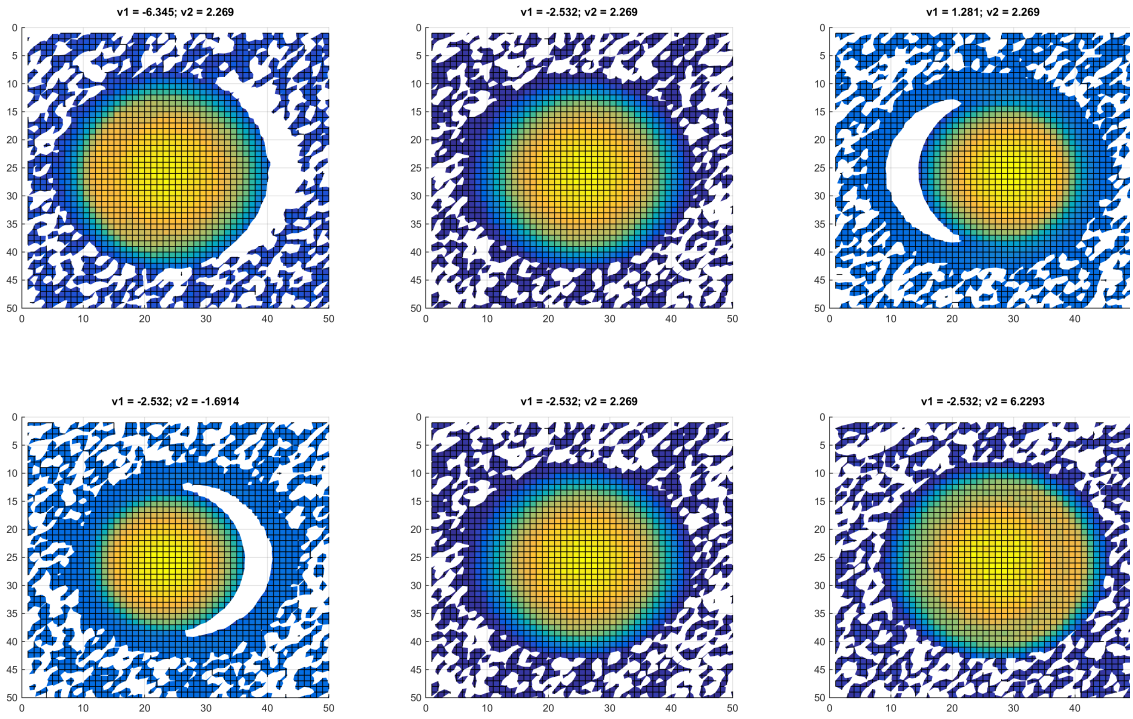

Figure 4.4: Example of Regular NLPCA Reconstructions of Simulated Point Cloud Data for Constant v_2, Increasing v_1 (Top, Left to Right) and Constant v_1, Increasing v_2 (Bottom, Left to Right)

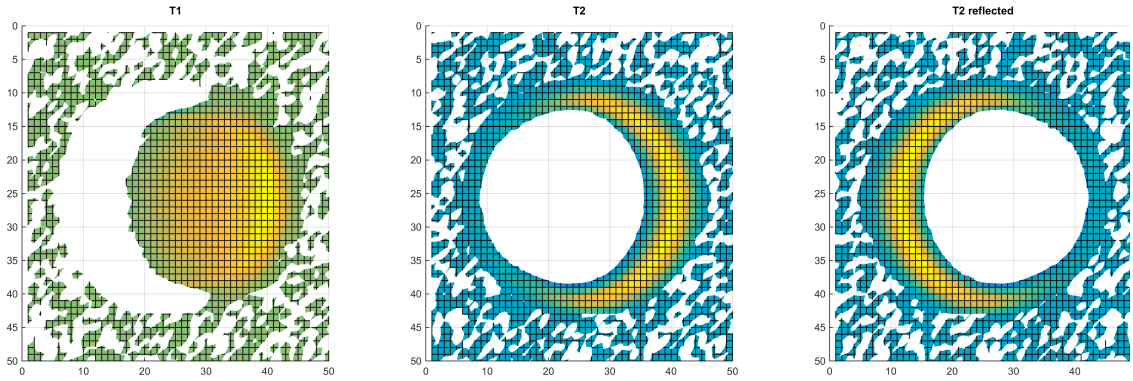

Figure 4.5: Tangent Vectors From a Regular NLPCA Solution Evaluated at a Single Point

be used in other applications where similar prior knowledge exists, or $TVCS_1$ could simply be used to approximate the measure if no prior knowledge is available. Note that the prior knowledge is only used for calculating the $TVCS$ metric that we use to numerically compare our ANLPCA method to regular NLPCA with respect to the separation of the variation patterns; this information is not used by the ANLPCA method, making it applicable to scenarios where no prior information about the variation sources is known.

4.4 Experimental Results

4.4.1 Simulated Point Cloud Data

The simulated point cloud dataset was described in the previous section and examples of the data are provided in Figures 4.2 and 4.4. The dataset consisted of 12,000 training instances and 4,200 testing instances, with both sets of data simulated using the same two variation patterns with additive Gaussian noise. During backpropagation training, 30% of the training instances were reserved for a validation set and the model was trained on the remaining 70% of the 12,000 observations. The 4,200 test instances were only used to calculate the $TVCS$ measure and the average SSE after training was complete. The average SSE is defined as

$$\text{Average SSE} = \frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{M}(x_{i,m} - \hat{x}_{i,m})^2 \tag{4.8}$$

Figure 4.2 depicts a typical ANLPCA solution obtained on this dataset, which has the desired separation of the variation patterns between v_1 and v_2. The model depicted in this figure had an average SSE of 1.167 and a $TVCS$ value of 0.095. It was trained for 200 epochs using batch backpropagation with a batch size of 500 observations. The model depicted in Figure 4.4 was trained using the same parameters and randomly initiated starting weights, but with regular NLPCA for 100 epochs

rather than ANLPCA. This resulted in a $TVCS$ value of 0.911 and an average SSE of 0.944. Although NLPCA yielded a lower SSE solution with less training, it completely mixes the variation patterns between the bottleneck nodes and therefore fails at the objective of identifying the distinct variation patterns.

To further compare ANLPCA and regular NLPCA, we trained models using each method starting from 30 different randomly initiated starting weights. Each model used the same network structure depicted in Figure 4.1 with two bottleneck nodes and a single hidden layer on each side of the bottleneck layer. As in the previous example, we trained the regular NLPCA models for 100 epochs and the ANLPCA models for 200 epochs because the alternating method typically yields slower learning. We also trained a third set of 30 models using 500 epochs of ANLPCA to obtain results that have comparable average SSE values to the regular NLPCA solution.

Other than the training method and the number of epochs, all other parameters were the same across the three sets of trained models. The standard neural network model parameters, including the number of hidden nodes in the first and third layers, were determined by performing an extensive search over different possible combinations and selecting the set which minimized the average SSE of reconstruction on a validation data set. The hidden layers on each side of the bottleneck layer had $n_1 = n_3 = 25$ nodes, yielding a complete network specification of 2500-25-2-25-2500 for the 2500-dimensional inputs, 25 nodes in each hidden layer, and 2 bottleneck nodes.

Both sets of ANLPCA models used an alternating frequency of $R = 25$ for the extra objective function term $J_r(\mathbf{W})$, which was chosen by monitoring the reduction in the average SSE by training epoch. This typically reveals an 'elbow' in the curve indicating where it is appropriate to alternate the penalty and begin training the other bottleneck node. An example of this where an alternating frequency of $R = 10$ epochs

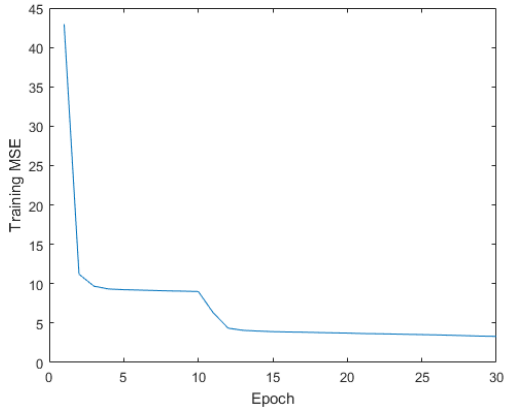

Figure 4.6: Example of the Reduction in Reconstruction Error by Training Epoch for a 2500-25-2-25-2500 ANLPCA Model With $R = 10$

was used is provided in Figure 4.6. A steep reduction in the MSE of reconstruction early in training is followed by smaller reductions in subsequent epochs after the model has learned as much information as possible using only a single bottleneck node, which is clearly identified by the 'elbow' in the plot. Another steep reduction in the error occurs when the penalty has alternated after epoch 10, corresponding to learning the second variation pattern while updating the weights for the other bottleneck node. The alternating frequency parameter R can be chosen by monitoring this reduction in the reconstruction MSE during training and alternating the penalty after it is clear that the model has passed the 'elbow' point in the curve. Alternatively, cross validation could be used to choose the value of R which minimizes the TVCS of the resulting model.

The $TVCS$ values for the models trained on the simulated point cloud data are summarized in Figure 4.7. Each boxplot shows the distribution of $TVCS$ measured across the 30 models trained using each learning method. ANLPCA with both 200 and 500 epochs produced substantially smaller $TVCS$ values than regular NLPCA, and visualization of each of the trained ANLPCA models revealed that the v_1 and v_2 bottleneck nodes each modeled distinct variation patterns similar to the solution

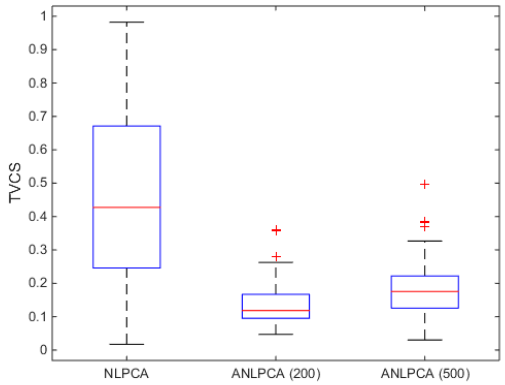

Figure 4.7: *TVCS* Measure Evaluated Across 30 Different Randomly Initiated Weights for 2500-25-2-25-2500 Networks Trained With NLPCA, ANLPCA Trained for 200 Epochs, and ANLPCA Trained for 500 Epochs

depicted in Figure 4.2. Conversely, many of the models trained with regular NLPCA yielded solutions similar to that depicted in Figure 4.4 where v_1 and v_2 each modeled a combination of the patterns. This is reflected in the much larger values of $TVCS$ observed for regular NLPCA.

Figure 4.8 shows the distribution of average SSE values across the 30 models trained on each method. Regular NLPCA with only 100 epochs outperforms ANLPCA with 200 epochs based on average SSE, although both solutions were considered acceptable from the perspective of having the reconstructed pattern resemble the originally observed pattern. Increasing the number of ANLPCA epochs to 500 provides average SSE values that are as good or better than the NLPCA 100 epoch solution with only a modest increase in the $TVCS$ measure. Figures 4.7 and 4.8 demonstrate how ANLPCA avoids undesirable solutions where the variation patterns are mixed between the bottleneck nodes, but also requires more training epochs to achieve the same level of average SSE as regular NLPCA. Thus, for a fixed amount of training epochs, there is a trade-off between interpretability and the accuracy of the reconstruction produced by the model.

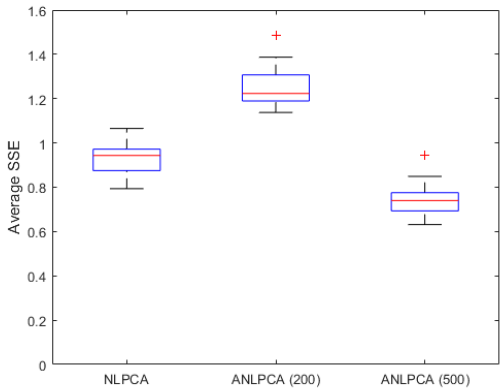

Figure 4.8: Average SSE Evaluated Across 30 Different Randomly Initiated Weights for 2500-25-2-25-2500 Networks Trained With NLPCA, ANLPCA Trained for 200 Epochs, and ANLPCA Trained for 500 Epochs

4.4.2 MNIST Handwritten Digits Database

The second set of data that we used to compare ANLPCA to NLPCA is a subset of the MNIST handwritten digits database (LeCun and Cortes, 1998). Specifically, we use the 6,131 images of the digit 3 contained in this database, of which 20% of the observations are reserved for testing. The remaining 80% of the images are used for training data, with a portion (10%) of the training data used as a validation set during backpropagation.

Learning for both ANLPCA and NLPCA was done via batch backpropagation with 750 observations in each batch, with the model parameters selected by performing a parameter search over different combinations and choosing the set which minimized the reconstruction error on validation data. These parameters included the number of nodes in the hidden layers, the coefficient of the weight decay term used in the objective function, the momentum value, and the standard deviation of the mean-zero normal distribution used to randomly initialize the weights. All models had the network structure depicted in Figure 4.1 with a single hidden layer on each side of the bottleneck layer. NLPCA models were trained for 100 epochs and

ANLPCA models were trained for 200 epochs because only the weights for one of the two bottleneck nodes are primarily learned during a given epoch of ANLPCA training. We used an alternating frequency of $R = 50$ for the ANLPCA training, which was again chosen by monitoring the change in the reconstruction SSE by training epoch and identifying the 'elbow' in the curve.

Two nodes were used in the bottleneck layer for all of the trained models. To measure the extent to which each bottleneck node characterizes a distinct variation pattern, we use only $TVCS_1$ rather than taking the maximum of $TVCS_1$ and $TVCS_2$ as with the previous example because we do not have any prior knowledge on the nature of the variation patterns contained in this dataset. The input and output layers have $M = 784$ nodes corresponding to the dimensionality of the MNIST data, and we use $n_1 = n_3 = 150$ nodes in the hidden layers on either side of the bottleneck layer where this number of hidden nodes was determined by performing a parameter search using validation data. The models were therefore specified as 784-150-2-150-784 networks for the 784-dimensional inputs, 150 nodes in each hidden layer, and 2 nodes in the bottleneck layer.

An example of the solution obtained by ANLPCA on the MNIST data is presented in Figure 4.9. The first bottleneck node v_1 appears to control the rotation of the digit while the second bottleneck node v_2 characterizes the curvature of the bottom of the digit. This solution had an average SSE of 33.98 and a $TVCS_1$ value of 0.039. The NLPCA solution for the same set of initial starting weights is depicted in Figure 4.10. This model had an average SSE of 31.87 and a $TVCS_1$ value of 0.67. In contrast to the ANLPCA model, the NLPCA model appears to characterize both the rotation of the digit and the bottom curvature with both v_1 and v_2, which is reflected in its large $TVCS_1$ score.

As with the simulated point cloud data, we trained models with both NLPCA

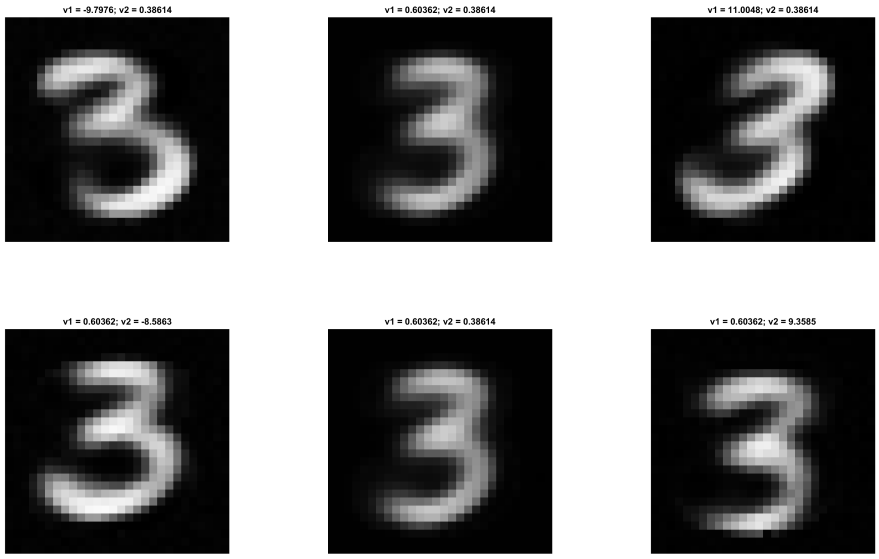

Figure 4.9: Example of 784-150-2-150-784 ANLPCA Network Reconstructions of MNIST Data for Constant v_2, Increasing v_1 (Top, Left to Right) and Constant v_1, Increasing v_2 (Bottom, Left to Right)

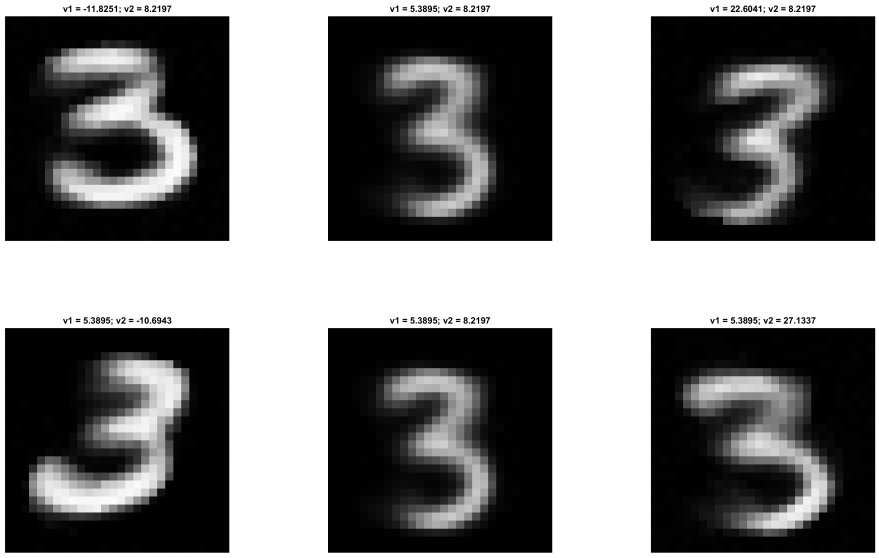

Figure 4.10: Example of 784-150-2-150-784 NLPCA Network Reconstructions of MNIST Data for Constant v_2, Increasing v_1 (Top, Left to Right) and Constant v_1, Increasing v_2 (Bottom, Left to Right)

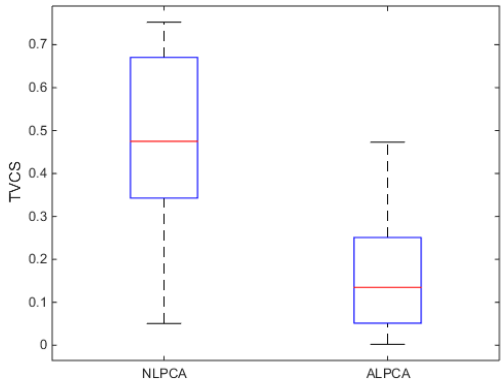

Figure 4.11: $TVCS_1$ Measure Evaluated Across 30 Different Randomly Initiated Weights for 784-150-2-150-784 Networks Trained With NLPCA and ANLPCA Using the MNIST Handwritten Digits

and ANLPCA using 30 different randomly initiated starting weights for the network to compare their performance. A boxplot comparing their $TVCS_1$ values across the 30 trials is provided in Figure 4.11. The ANLPCA models consistently produced lower $TVCS_1$ scores, indicating that the ANLPCA method results in solutions which better separate the directions represented by the bottleneck nodes. A comparison of the average SSE is provided in Figure 4.12. Similar to the previous example, ANLPCA provides lower $TVCS_1$ while yielding solutions with slightly higher average SSE. However, the examples depicted in Figure 4.9 show that the digit is still easily recognizable at this level of reconstruction error. In order to obtain more distinct patterns, ANLPCA might be considered to be a constrained solution, so that a slight degradation of the fit can be expected relative to NLPCA for the same amount of training.

4.5 Conclusion

In this paper, we addressed the difficult problem of learning distinct nonlinear sources of variation in high-dimensional data. Our strategy to promote the learning

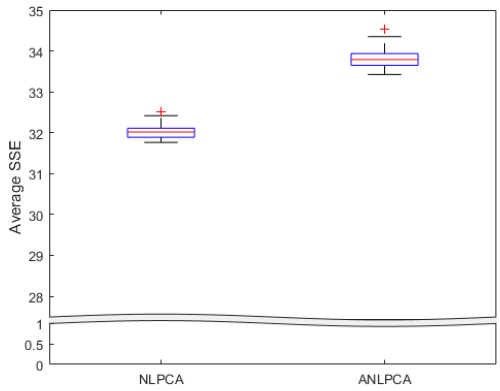

Figure 4.12: Average SSE Evaluated Across 30 Different Randomly Initiated Weights for 784-150-2-150-784 Networks Trained With NLPCA and ANLPCA Using the MNIST Handwritten Digits

of distinct sources of variation is conceptually simple and easily extends from more traditional methods. The previous literature has not provided a methodology for this important problem, and we expect this initial work to be enhanced with alternative approaches. Our work facilitates efforts to visualize independent variation patterns, which can be applied to a wide-range of exploratory data analysis problems.

Our results using simulated point cloud data and the MNIST handwritten digits data show that the ANLPCA method consistently produces results with the desired separation of distinct variation patterns. We introduced the $TVCS$ metric, which is motivated by the orthogonality of loadings in PCA and serves as an effective measure that can be used to quantify this distinctness characteristic of the solution. This metric will facilitate the comparison of our ANLPCA method to future work on learning distinct nonlinear variation patterns.

In future research, we would like to explore alternative training methods that can be introduced to encourage distinctness of the variation patterns in ANN models. For example, constraints on the properties of an ANN could be implemented via regularization penalties in order to encourage the true variation sources to be learned

by different bottleneck nodes in the model. A combination of our proposed ANLPCA method and this regularized approach could potentially offer improvements over the results presented in this work. Finally, we would like to explore the effect that differing relative magnitudes of nonlinear variation sources have on the effectiveness of our ANLPCA methodology.

Chapter 5

REGULARIZED LEARNING FOR DISTINCT FEATURE DISCOVERY USING AUTOASSOCIATIVE NEURAL NETWORKS

5.1 Introduction

Dimensionality reduction is an important task in many problem domains where it is desirable to understand variation in a system described by a high-dimensional feature space. Often the intrinsic dimensionality of the data characterizing its variation is much smaller than the dimensionality of the observed data, which makes the identification of a good low-dimensional representation critical to understanding the complexity of the system. While typically the quality of a low-dimensional representation is measured by how well the original source data can be reconstructed from it, an equally important objective in the analysis of complex high-dimensional data is how well the solution can be interpreted with respect to the variation exhibited in the original feature space. This is demonstrated by the popularity of Principal Component Analysis (PCA) as a method for summarizing data. Because the principal components are constrained to represent orthogonal directions in the original feature space, retaining only the top components in a PCA solution can provide an effective low-dimensional representation where each of the components characterizes a source of variation in the original data.

PCA is limited to finding low-dimensional representations where the components represent linear combinations of the original features. Thus, it is not suitable for discovering variation sources which represent nonlinear functions of the feature space. A nonlinear extension of PCA previously proposed is the use of autoassociative neu-

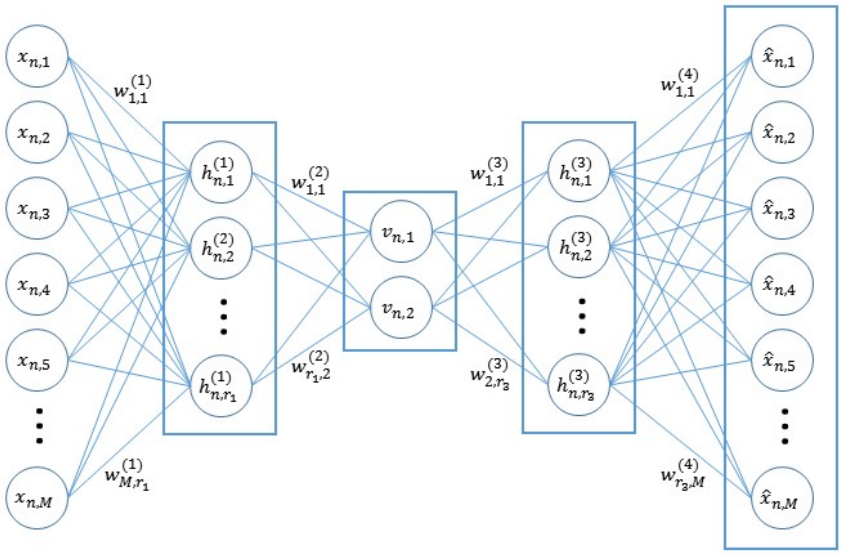

Figure 5.1: Network Architecture for an ANN Model With $P = 2$ Components

ral networks (ANNs) (Kramer, 1991) which are neural network models trained to reproduce their input after passing through a 'bottleneck' layer forcing information compression. An example of a simple ANN model structure is provided in Figure 5.1 where the bottleneck layer in the middle of the network has only 2 nodes in order to force learning of a 2-dimensional representation of the M-dimensional input data. These ANN models are typically trained iteratively via gradient descent algorithms with the objective of minimizing the error of reconstructing the original input data at the output layer of the network, which has the same dimensionality as the input data. Existing ANN approaches for nonlinear dimensionality reduction do not enforce any constraints on the solution to encourage learning of interpretable solutions, unlike the orthogonality constraints on a PCA solution. Thus, the nonlinear 'components' learned by an ANN model (which are represented by the values of the nodes in the bottleneck layer) often align with the same sources of variation early in training (Kramer, 1991) and produce solutions where each of the components represent a mixture of multiple variation sources.

In order to avoid this deficiency, we propose a new regularized learning framework which is motivated in-part by the properties of features extracted from linear methods such as PCA. Our approach leverages the parametric model representation provided by an ANN to control characteristics such as the correlation between extracted features, the kurtosis of the distribution of values for each feature, and the orientation of the tangent vectors to the manifold defined by each of the learned features. This allows for the extension of desirable solution properties provided by linear feature extraction methods to the nonlinear case, such as uncorrelated components and orthogonal loadings in PCA. More generally, our proposed framework of imposing constraints on ANNs to improve the interpretability of the extracted features can be applied to a variety of deep learning problems in which it is desirable to understand the features being learned by the model.

As will be discussed in Section 5.3, our task of blindly learning nonlinear variation sources is a difficult one with no single solution to address all problems. In practice, there could be many different properties that would be desirable to enforce on an ANN model through regularization depending on the problem at hand. We are proposing just three regularization terms in this work, but our greater contribution is demonstrating how controlling properties of an ANN model through regularized training improves the interpretability of the learned features.

The remainder of this paper is organized as follows. In Section 5.2, we provide a mathematical description of the problem and introduce terminology that will be used throughout the rest of the paper. Section 5.3 discusses relevant previous work on the problem of identifying interpretable low-dimensional representations of high-dimensional data. Our methodology for regularized ANN learning is presented in Section 5.4, and experimental results demonstrating the effectiveness of our approach are detailed in Section 5.5. A summary of our contributions and concluding remarks

are provided in Section 5.6.

5.2 Problem Description

The multivariate data analyzed in this work is assumed to have the functional form $\mathbf{x}_n = \mathbf{f}(\mathbf{s}_n) + \mathbf{w}_n$, where $\mathbf{x}_n$ is the n^{th} vector instance in a set of N data observations. The data instances exist in an M-dimensional space where each component of an instance is represented notationally as $\mathbf{x}_n = [x_{n,1}, x_{n,2}, \ldots, x_{n,M}]$. A given instance $\mathbf{x}_n$ is assumed to arise from P unobserved sources of variation $\mathbf{s}_n = [s_{n,1}, s_{n,2}, \ldots, s_{n,P}]$, which are mapped to the M-dimensional space of the observed data via the M common nonlinear functions $\mathbf{f}(\mathbf{s}_n) = [f_1(\mathbf{s}_n), f_2(\mathbf{s}_n), \ldots, f_M(\mathbf{s}_n)]^T : \mathbb{R}^P \mapsto \mathbb{R}^M$. The P sources of variation are assumed to be independent, which is a common assumption for variation source discovery problems (Shi *et al.*, 2016). The observed data instances are also corrupted by an additive noise component $\mathbf{w}_n$.

Our objective is to identify and visualize the true underlying variation sources $\mathbf{s}_n$ using only the observed data instances $\mathbf{x}_n$, $n \in \{1, \ldots, N\}$. The model we use for this task is an ANN which is trained to reconstruct its M-dimensional input at the output layer after passing through a P-dimensional bottleneck layer in the middle of the network. The basic model structure is depicted in Figure 5.1 for the case where $P = 2$. The j^{th} node in the k^{th} hidden layer of the network for a given instance n is denoted as $h_{n,j}^{(k)}$ and there are a total of r_k nodes in the k^{th} layer. The hidden nodes in the bottleneck layer at the middle of the network are denoted $\mathbf{v}_n = [v_{n,1}, \ldots, v_{n,P}]$ and represent the values of the unobserved variation patterns learned by the network (i.e. the model's approximation of $\mathbf{s}_n$); these nodes in the bottleneck layer will be subsequently referred to as bottleneck nodes or nonlinear components (analogous to the principal components in a PCA solution).

The arcs in Figure 5.1 represent weights connecting nodes in a given layer of

the network to nodes in the next layer. We let $w_{ij}^{(k)}$ denote the value of the weight connecting the i^{th} node of layer $k-1$ to the j^{th} node of layer k. The entire set of weights for the network, $\mathbf{W}$, along with the activation functions for each of the nodes in the network, fully specify a trained ANN model. The portion of the network consisting of the bottleneck layer through the output layer represents a 'decoder' which can be used to visualize the learned variation patterns by reconstructing different output vectors $\hat{\mathbf{x}}$ for varying values of $\mathbf{v}$. We will refer to the weights and activation functions defining this decoder portion of the network because they provide a mapping of the P-dimensional learned variation sources to the M-dimensional space of the observed data.

The bottleneck nodes in an ANN model define a P-dimensional manifold existing in the M-dimensional space of the observed data. The P bottleneck node values for a given data instance represent the coordinates of the instance in the manifold and the decoder portion of the network defines a mapping of these manifold coordinates to the M-dimensional observed data space. Because ANNs provide this functional representation of the manifold, we can control the structure of the learned manifold by imposing constraints on the functional relationships during model training. This represents a key advantage of ANN models over alternative non-parametric approaches to manifold learning and motivates our use of regularization terms to learn more interpretable nonlinear features.

5.3 Previous Work

Kernel PCA (KPCA) has been presented as an extension of linear which maps the data vectors $\mathbf{x}$ in M-dimensional space to some Q-dimensional ($Q >> M$ typically, and Q may be infinite) feature space via map $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_Q(\mathbf{x})]^T$, where each $\phi(\mathbf{x})$ is some scalar-valued nonlinear function of $\mathbf{x}$. PCA is then performed in the

Q-dimensional space of the feature vectors $\{\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_Q(\mathbf{x})\}$. To avoid the computational complexity of performing calculations in the high-dimensional space of these feature vectors, the mapping is defined implicitly using kernel functions (Schölkopf *et al.*, 1998). KPCA provides some denoising capabilities and it has been applied to variation pattern analysis previously (Sahu *et al.*, 2014; Shinde *et al.*, 2014; Im *et al.*, 2012). However, it is not suited for our task of visualizing the variation sources because KPCA solutions do not provide a parametric representation of the learned patterns.

Independent Component Analysis (ICA) is one method commonly used for blind source separation (BSS), which is related to our task of identifying interpretable variation patterns. ICA identifies solution which minimizes the statistical dependence between the components (Comon, 1994). Traditional ICA methods are only suitable for cases where the variation patterns are linear in nature, making them inappropriate for our task of identifying nonlinear patterns. Alternative BSS methods for linear variation patterns have been proposed with applications to manufacturing variability analysis in (Shan and Apley, 2008) and (Apley and Lee, 2003), but these methods do not address nonlinear variation pattern discovery. Unlike the case of linear patterns, nonlinear ICA solutions are nonunique and differ greatly from the solution to the much harder nonlinear BSS problem because any arbitrary function of a pair of independent random variables will also be independent. (Jutten and Karhunen, 2003). A nonlinear ICA solution is insufficient for our task of visualizing the true nonlinear variation sources because the original variation patterns can be mixed among the extracted features while still being statistically independent. Additionally, many existing nonlinear BSS methods focus on extracting the original sources whereas we also want to learn the nonlinear functions $\mathbf{f}(\mathbf{s}_n)$ which map the sources back to the original feature space for the purpose of visualization.

As discussed in Section 5.1, ANNs have been proposed as a form of nonlinear PCA (NLPCA) and constitute the modeling framework used in this paper. Several variants of ANNs have been proposed previously. Circular PCA implements an ANN with the constraint that the bottleneck nodes must fall on the unit circle, allowing a pair of bottleneck nodes to be described by a single angle parameter (Kirby and Miranda, 1996). Inverse NLPCA is a variant of ANNs which learns only the decoding portion of the network, treating the bottleneck nodes as model parameters learned during backpropagation along with the weights (Scholz *et al.*, 2008). Hierarchical nonlinear principal component networks (HNPCN) seeks a solution where the nonlinear components are hierarchically ordered (Saegusa *et al.*, 2004; Scholz and Vigário, 2002). ANN models with many hidden layers between the bottleneck layer and the input/output layers have been proposed and are commonly referred to as deep autoencoders (Hinton and Salakhutdinov, 2006). The difficulty of learning deep network architectures is mitigated by use of unsupervised pre-training prior to performing backpropagation. Pre-trained deep autoencoders were applied to learning efficient low-dimensional representations of nonlinear manufacturing data, outperforming alternative methods based on re-indexing and manifold learning (Howard *et al.*, 2015). However, none of these methods specifically address the issue of separating the true variation patterns among the learned bottleneck nodes in an ANN.

Multiple regularized learning methods for autoencoders have been proposed to achieve various modeling objectives in addition to minimizing the reconstruction error. Contractive autoencoders encourage robustness of the low-dimensional representation learned by the model to changes in the input by penalizing the partial derivatives of the bottleneck nodes with respect to each of the inputs (Rifai *et al.*, 2011). Denoising autoencoders attempt to learn data representations which are robust to corruption of the input data by adding noise to the data instances presented

at the input layer of the network and then evaluating the reconstruction error using the uncorrupted data instances (Vincent *et al.*, 2008). Contractive and denoising autoencoders have been shown to be closely related; whereas contractive autoencoders penalize the partial derivatives of the bottleneck layer units with respect to the input layer units, denoising autoencoders can be interpreted as penalizing the output layer nodes with respect to the input layer nodes (Alain and Bengio, 2014). While our proposed methodology also represents a type of regularized autoencoder, the purpose of our regularization method differs from previous work in that we attempt to learn representations where each of the bottleneck nodes represents a unique variation pattern impacting the observed data.

Alternating Nonlinear Principal Component Analysis (ANLPCA) is a method for training ANNs with the objective to separate the true variation sources among the learned nonlinear components (Howard *et al.*, 2016a). ANLPCA uses an alternating penalty which constrains updates to the weights connected to only a single bottleneck node during any given epoch of backpropagation training, which encourages the true variation sources to load onto only a single nonlinear component. The authors also introduced the Tangent Vector Cosine Similarity (TVCS) measure for evaluating how well an ANN model has separated true variation sources among the learned bottleneck nodes. The effectiveness of TVCS as a performance measure for our objective of identifying the true sources of variation motivates our use of TVCS as one of the proposed regularization terms in this work. Although ANLPCA also aims to produce ANN solutions which are interpretable, this prior work represents an initial approach to the problem whereas the methodology we present here is an enhanced solution building upon the existing body of work on regularized learning.

5.4 Methodology

A common method for training ANN models is to iteratively minimize an objective function consisting of the sum of squared error (SSE) using a method such as stochastic gradient descent. For a set of N training instances and a given model defined by the weights $\mathbf{W}$, the SSE (denoted as $J_E(\mathbf{W})$) is defined in Equation 5.1.

$$J_E(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} (x_{n,m} - \hat{x}_{n,m})^2 \tag{5.1}$$

Our regularized learning approach for ANNs adds a set of regularization terms to the objective function in order to penalize deviation of the model from several constraints. The three regularization terms we propose penalize the TVCS of the model (denoted $J_T(\mathbf{W})$), the pairwise correlation between the bottleneck nodes (denoted $J_C(\mathbf{W})$), and the deviation of kurtosis from that of the uniform distribution for each bottleneck node (denoted $J_K(\mathbf{W})$). These three regularization terms will be detailed in the subsequent subsections. Letting λ_T, λ_C, and λ_K denote user-defined parameters controlling the weight of each regularization term, the complete objective function minimized during stochastic gradient descent training is

$$J(\mathbf{W}) = J_E(\mathbf{W}) + \lambda_T J_T(\mathbf{W}) + \lambda_C J_C(\mathbf{W}) + \lambda_K J_K(\mathbf{W}) \tag{5.2}$$

The purpose of the regularization terms in Equation 5.2 is to encourage the learning of nonlinear features which have desirable properties for the purpose of interpretation and visualization. Without regularization, the typical method for training ANNs by minimizing SSE only encourages the learned features to provide an efficient representation of the original data in the low-dimensional encodings of the bottleneck nodes. The flexibility in modeling complex nonlinear functions provided by an ANN allows for multiple different functional relationships to be learned by the network in arriving at an efficient low-dimensional representation. In regularizing the objective

function, we aim to restrict the set of possible functional relationships which can be learned to those accurately representing the true sources of variation governing the observed data.

5.4.1 TVCS Regularization

The TVCS metric described previously in Section 5.3 measures the orthogonality of tangents to the manifold defined by an ANN and is motivated by the condition of orthogonal loadings in a PCA solution. A manifold is defined by the functional relationship between the bottleneck node values $\mathbf{v}_n$ and the reconstructions at the output layer $\hat{\mathbf{x}} = \hat{\mathbf{f}}(\mathbf{v}_n)$, where the function $\hat{\mathbf{f}}(\ ) : P \rightarrow M$ represents the decoder portion of the network (i.e. the model's approximation of the true nonlinear functions $\mathbf{f}(\ )$). The manifold is embedded in $\mathbb{R}^{\mathrm{M}}$ and at a given point $\mathbf{v}_n$, there exists an M-dimensional tangent vector $\mathbf{T}_n^{(p)}$ to the manifold for each of the $p \in \{1, \ldots, P\}$ bottleneck nodes. A tangent vector $\mathbf{T}_n^{(p)}$ is the partial derivative of $\hat{\mathbf{f}}(\mathbf{v}_n)$ with respect to the p^{th} bottleneck node $v_{n,p}$ and evaluated at data instance n, defined as

$$\mathbf{T}_n^{(p)} = \frac{\partial\ \hat{\mathbf{f}}(\mathbf{v}_n)}{\partial\ v_{n,p}} \tag{5.3}$$

for $p \in \{1, \ldots, P\}$ and $n \in \{1, \ldots, N\}$. $\mathbf{T}_n^{(p)}$ approximates the path in the manifold generated by changes in $v_{n,p}$ while holding the other bottleneck nodes constant. For the purpose of interpreting independent variation sources, the tangent vectors are important because they provide local characterizations of the variation sources learned by each of the bottleneck nodes. Although there exists a P-dimensional tangent space at each point on the manifold, only the tangent vectors $\mathbf{T}_n^{(p)}$ corresponding to the bottleneck nodes $p \in \{1, \ldots, P\}$ are relevant for interpreting the assumed independent variation sources because movement along other directions in the tangent space would require simultaneous changes to more than one of the bottleneck nodes.

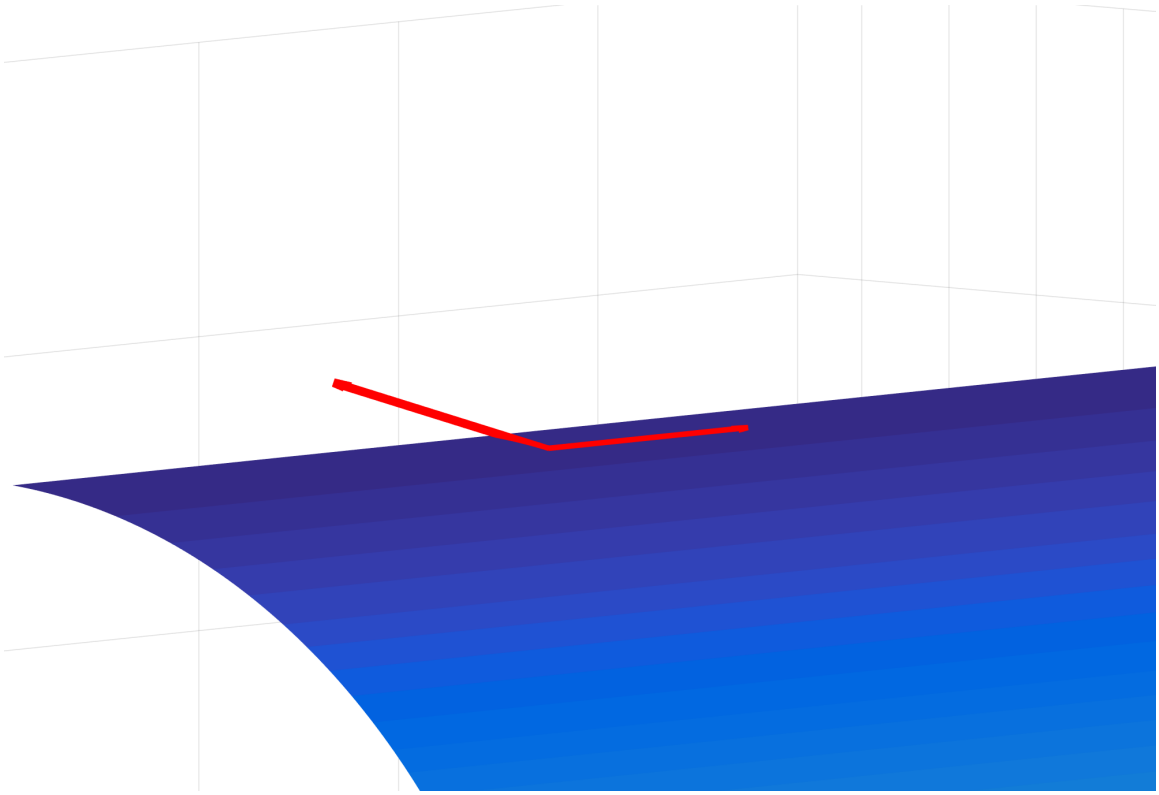

Figure 5.2: Example of a Pair of Tangent Vectors (Red) at a Single Point on a Two-Dimensional Manifold Existing in Three-Dimensional Space

An illustration of two tangents vectors at a single point on a manifold is provided in Figure 5.2. In the figure, a two-dimensional manifold existing in three-dimensional space is depicted as a blue surface. The tangent vectors approximating the path in the manifold generated by two independent variation sources at a single point are plotted as red lines in the figure. Note that the tangent vectors are orthogonal and point in different directions of the three-dimensional space. This figure depicts the pair of tangent vectors at just a single point on the manifold, but there exist similar tangent vectors at each possible reconstructed point generated by an ANN model.

Our proposed TVCS regularization term measures the cosine of the angle between each pair of the P tangent vectors and penalizes deviations from zero in absolute value in order to encourage solutions where the tangent vectors are orthogonal. To simplify

notation, we now restrict our discussion to the case where $P = 2$ bottleneck nodes are included in the ANN. However, the methodology discussed here can be applied to cases where $P > 2$ nonlinear components are sought by evaluating the formulation on the $\frac{P(P-1)}{2}$ possible pairs of tangent vectors and averaging the resulting values. To compute our TVCS regularization term $J_T(\mathbf{W})$ for the case of $P = 2$ bottleneck nodes, we calculate the tangent vectors $\mathbf{T}_n^{(1)}$ and $\mathbf{T}_n^{(2)}$ corresponding to the two bottleneck nodes at each data instance $n \in \{1, \ldots, N\}$ and then set TVCS equal to the square of the cosine similarity of these vectors averaged over the data set, as follows.

$$J_T(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\mathbf{T}_n^{(1)} \cdot \mathbf{T}_n^{(2)}}{||\mathbf{T}_n^{(1)}||\ ||\mathbf{T}_n^{(2)}||} \right)^2 \tag{5.4}$$

Because the decoder portion of the network provides the functional relationship between the reconstructions $\hat{\mathbf{x}}$ and the bottleneck node values $\mathbf{v}$, exact analytical expressions for $J_T(\mathbf{W})$ are possible. For the basic ANN model structure depicted in Figure 5.1 with $P = 2$ nonlinear components and one hidden layer on each side of the bottleneck layer, the m^{th} component of the tangent vector $\mathbf{T}_n^{(1)}$ evaluated at data instance n, denoted $T_{n,m}^{(1)}$, when sigmoid activation functions are used for the third hidden layer and linear activation functions for the output layer is

$$T_{n,m}^{(1)} = \sum_{j=1}^{r_3} w_{jm}^{(4)} \cdot h_{n,j}^{(3)}(1 - h_{n,j}^{(3)}) \cdot w_{1j}^{(3)} \tag{5.5}$$

Similarly, the m^{th} component of the tangent vector $\mathbf{T}_n^{(2)}$ evaluated at data instance n is denoted as $T_{n,m}^{(2)}$ and equals

$$T_{n,m}^{(2)} = \sum_{j=1}^{r_3} w_{jm}^{(4)} \cdot h_{n,j}^{(3)}(1 - h_{n,j}^{(3)}) \cdot w_{2j}^{(3)} \tag{5.6}$$

Using these expressions for the tangent vectors defined in Equations 5.5 and 5.6, the exact value of our TVCS regularization penalty $J_T(\mathbf{W})$ is calculated as follows.

$$J_T(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^{N} \left[\frac{\sum_{m=1}^{M} \left(\sum_{j=1}^{r_3} w_{jm}^{(4)} \cdot h_{n,j}^{(3)}(1-h_{n,j}^{(3)}) \cdot w_{1j}^{(3)} \right) \cdot \left(\sum_{j=1}^{r_3} w_{jm}^{(4)} \cdot h_{n,j}^{(3)}(1-h_{n,j}^{(3)}) \cdot w_{2j}^{(3)} \right)}{\sqrt{\sum_{m=1}^{M} \left(\sum_{j=1}^{r_3} w_{jm}^{(4)} \cdot h_{n,j}^{(3)}(1-h_{n,j}^{(3)}) \cdot w_{1j}^{(3)} \right)^2} \sqrt{\sum_{m=1}^{M} \left(\sum_{j=1}^{r_3} w_{jm}^{(4)} \cdot h_{n,j}^{(3)}(1-h_{n,j}^{(3)}) \cdot w_{2j}^{(3)} \right)^2}} \right]^2 \tag{5.7}$$

5.4.2 Correlation Regularization

Another desirable property of PCA solutions is that the principal components are linearly uncorrelated. We can extend this condition to ANN dimensionality reduction by penalizing the square of the correlation coefficient evaluated on the bottleneck nodes in the middle layer of the network. For notational clarity, we again restrict attention to the special case of $P = 2$ bottleneck nodes, but the formulation can be extended to $P > 2$ by averaging the regularization term evaluated on each pair of the P bottleneck nodes. Letting $\bar{v}_1$ and $\bar{v}_2$ denote the mean values of the first and second bottleneck nodes, our proposed regularization term $J_C(\mathbf{W})$ to encourage the learning of uncorrelated features is

$$J_C(\mathbf{W}) = \left(\frac{\sum_{n=1}^{N} (v_{n,1} - \bar{v}_1)(v_{n,2} - \bar{v}_2)}{\sqrt{\sum_{n=1}^{N} (v_{n,1} - \bar{v}_1)^2} \sqrt{\sum_{n=1}^{N} (v_{n,2} - \bar{v}_2)^2}} \right)^2 \tag{5.8}$$

Similar to our TVCS regularization term, the correlation coefficient also has a geometric interpretation: a value of $J_C(\mathbf{W}) = 0$ implies that the two bottleneck node data vectors $[v_{1,1}, v_{2,1}, \ldots, v_{N,1}]$ and $[v_{1,2}, v_{2,2}, \ldots, v_{N,2}]$ are orthogonal after centering (Rodgers *et al.*, 1984). Thus, our $J_T(\mathbf{W})$ and $J_C(\mathbf{W})$ regularization terms can be viewed as enforcing orthogonality constraints on two different aspects of an ANN model: $J_T(\mathbf{W})$ encourages orthogonality of the linear approximations to the decoder portion of the network while $J_C(\mathbf{W})$ penalizes non-orthogonality of the centered bottleneck node data vectors, which are a function of the encoder portion of the network.

5.4.3 Uniform Excess Kurtosis Regularization

Our third proposed regularization term measures the kurtosis of the distribution of values for each bottleneck node and penalizes the deviation of these quantities from the kurtosis of the uniform distribution, which we refer to as the uniform excess

kurtosis. This term is analogous to the concept of excess kurtosis except that the kurtosis of the uniform distribution (1.8) is used as a baseline instead of the kurtosis of the normal distribution. Our uniform excess kurtosis regularization term $J_K(\mathbf{W})$ is the squared deviation from 1.8 of the kurtosis for each bottleneck node, summed across the P bottleneck nodes in the model. For the special case of $P = 2$ bottleneck nodes, $J_K(\mathbf{W})$ is defined as

$$J_K(\mathbf{W}) = \left(\frac{\frac{1}{N}\sum_{n=1}^{N}\left(v_{n,1} - \bar{v}_1\right)^4}{\left(\frac{1}{N}\sum_{n=1}^{N}\left(v_{n,1} - \bar{v}_1\right)^2\right)^2} - 1.8 \right)^2 + \left(\frac{\frac{1}{N}\sum_{n=1}^{N}\left(v_{n,2} - \bar{v}_2\right)^4}{\left(\frac{1}{N}\sum_{n=1}^{N}\left(v_{n,2} - \bar{v}_2\right)^2\right)^2} - 1.8 \right)^2 \tag{5.9}$$

The $J_K(\mathbf{W})$ regularization term encourages the learning of solutions where each bottleneck node is approximately uniformly distributed. This is a desirable property for learning independent variation sources because it prevents the density of the bottleneck node distribution from concentrating in certain regions where the bottleneck node can co-adapt with other bottleneck nodes. Although we do not make any distributional assumptions about the unknown variation sources $\mathbf{s}_n$, our empirical results presented in Section 5.5 have validated the usefulness of this property for avoiding ANN solutions where the bottleneck nodes have learned complex mixtures of the true variation sources. Our results also show that penalizing uniform excess kurtosis performs well even when the true variation sources $\mathbf{s}_n$ are not uniformly distributed.

5.5 Experimental Results

We tested our methodology using one simulated data set consisting of 3-dimensional data instances and one publicly-available data set consisting of real images. The simulated data set consists of data points sampled from a portion of the 3-dimensional swiss roll, which were generated by sampling over a range of two variation sources

and adding independent Gaussian noise to the result. The portion of the swiss roll used for this simulated data set was selected in order to simplify visualization of the tangent vectors to the manifold learned by an ANN model. The second data set consists of a portion of the MNIST handwritten digit images (LeCun and Cortes, 1998). For both data sets, we fit a set of ANN models having $P = 2$ nodes in the bottleneck layer both with and without our proposed regularized learning methodology. Because the solution obtained by an ANN depends on the random initialization of starting weights, we trained 30 models using each of the evaluated learning methods where each of the 30 models had starting weights initialized using a different random seed.

For the data sets used in our experiments, each dimension of the observed feature space contained values on a common scale of measurement. As such, we did not perform any preprocessing of the data prior to fitting the ANN models. However, application of our methodology to other problems may require scaling of the variables prior to fitting a model if the features are based on different measurement scales.

5.5.1 Parameter Selection

The parameter configurations used in our experiments were selected by performing a parameter search using a validation set withheld from the data. The parameters tuned using this method included the weight given to each of our proposed regularization terms in the objective function (λ_T, λ_C, and λ_K), the number of data instances randomly selected for a batch in each epoch of batch stochastic gradient descent, and the number of nodes in the first and third hidden layers (which were kept equal to make the model structure symmetric). Because we are interested in both minimizing the SSE of reconstruction and enforcing the constraints on the model defined by each of our regularization penalties, we selected the final set of parameters which achieved the minimum value of SSE on validation data while also having sufficiently

small values for the penalties $J_T(\mathbf{W})$, $J_C(\mathbf{W})$, and $J_K(\mathbf{W})$. Specifically, we specified threshold levels Φ_T, Φ_C, and Φ_K and chose the set of parameters which minimized the SSE of reconstruction while satisfying the conditions $J_T(\mathbf{W}) \leq \Phi_T$, $J_C(\mathbf{W}) \leq \Phi_C$, and $J_K(\mathbf{W}) \leq \Phi_K$ in the resulting model. We used threshold values $\Phi_T = \Phi_C = \Phi_K = 0.1$ in our parameter search to ensure that the regularization penalties were enforced to a reasonable level of precision.

The parameter search was conducted by specifying three different levels of the five parameters and then training models using each of the $3^5 = 243$ possible combinations of parameter configurations. If none of the resulting models satisfied the threshold requirements for the regularization penalties $J_T(\mathbf{W})$, $J_C(\mathbf{W})$, and $J_K(\mathbf{W})$, the search values for the parameters λ_T, λ_C, and λ_K were increased and the search was repeated over the new range of parameters. Similarly, if the minimum SSE models after a search concentrated at the ends of the range of searched values for a particular parameter (such as the smallest or largest number of hidden nodes considered), the range of values for that particular parameter was expanded in the direction of the optimal model configurations and another parameter search was performed. This methodology was used in order to perform an exhaustive search over large regions of the parameter configuration space without having to train an unreasonable number of models. In practice, simpler parameter search methods could be used (e.g. tuning only a single parameter at a time rather than considering all possible combinations of multiple parameters).

5.5.2 Swiss Roll Data Set

Simulated data is useful for evaluating the effectiveness of our method because it allows comparison of the true variation sources $\mathbf{s}_n$ and their nonlinear functions $\mathbf{f}(\mathbf{s}_n)$ to the approximations $\mathbf{v}_n$ and $\hat{\mathbf{f}}(\mathbf{v}_n)$ learned by an ANN model. To quantify the accu-

racy of an ANN model's representation of the true variation sources, we measure the size of the angle between each tangent vector in an ANN model and its corresponding tangent derived from the true nonlinear functions defining the simulated variation patterns. Specifically, we define the variation source similarity (VSS) metric based on the true variation source tangents $\mathbf{U}_n^{(p)}$ and $\mathbf{U}_n^{(q)}$ for $p, q \in \{1, \ldots, P\}$ as

$$VSS = \min\left\{ \frac{1}{PN} \sum_{p=1}^{P} \sum_{n=1}^{N} \arccos\left(\left| \frac{\mathbf{T}_n^{(p)} \cdot \mathbf{U}_n^{(q_p)}}{||\mathbf{T}_n^{(p)}||\ ||\mathbf{U}_n^{(q_p)}} \right| \right) \;\middle|\; q_p \in \{1, \ldots, P\},\ q_p \neq q_\ell\ \forall\ p \neq \ell \right\} \tag{5.10}$$

where

$$\mathbf{U}_n^{(q_p)} = \frac{\partial\ \mathbf{f}(\mathbf{s}_n)}{\partial\ s_{n,q_p}} \tag{5.11}$$

The minimum function used in Equation 5.10 is necessary because the ordering of the bottleneck nodes $\mathbf{v}_n^{(p)}$ learned by an ANN model will not necessary match the (arbitrary) ordering of the simulated variation sources $\mathbf{s}_n^{(p)}$ when the model has correctly learned the variation sources. VSS measures the degree of the angle between each of the ANN's tangent vectors and the true tangent vector to which it has been matched, averaged across the N data instances. A value of VSS near zero indicates that the ANN has correctly learned the variation sources simulated in the data wheres larger values of VSS provide an indication of some amount of 'mixing' of the true variation sources among each of the features learned by the model.

Simulated data also allows us to evaluate models on the ability to ignore noise and reconstruct the true noiseless data instance. For our simulated data results, we therefore fit the models using the noisy data instances but evaluated the SSE of reconstruction using the noiseless data instances which were available as a byproduct of simulation. This corresponds to replacing $x_{n,m}$ in the formulation of the SSE in Equation 5.1 with $f_m(\mathbf{s}_n)$ as defined in Section 5.2. The noiseless data would not be

available in practice and is used in our experiments only for evaluation purposes.

The first simulated data set for which we present results in this section was generated from a portion of the three-dimensional swiss roll. Each of the simulated data points exist in three-dimensional space and were simulated as a function of two underlying sources of variation. Thus, $M = 3$ and $P = 2$ for the ANN models fit to this data, which is advantageous for illustrative purposes because we can visualize both the manifold and the learned tangent vectors in three-dimensional space. The data was simulated by first generating a grid of equally-spaced values for $s_1 \in \left[\frac{3\pi}{2}, 2\pi\right]$ and $s_2 \in [0, 6]$. The noiseless three-dimensional coordinates were then obtained using the functional relationships $x_1 = s_1 \cos(s_1)$, $x_2 = s_2$, and $x_3 = s_1 \sin(s_1)$ evaluated at each point in the grid of s_1 and s_2 values. Independent Gaussian noise with mean zero and a standard deviation of 0.05 was added to the x_1, x_2, and x_3 dimensions to create the final observed data instances. The set of $N = 10,000$ data instances were generated twice to produce two data sets. One of the simulated data sets was used for training the models and the other set was withheld for testing, with the training and testing sets having their additive noise generated using different random seeds. A portion of the training set was used for validation in the parameter search experiments described previously.

A scatter plot of the $N = 10,000$ simulated data points prior to adding Gaussian noise is provided in Figure 5.3. The first variation pattern (s_1) controls the positioning of the point along the curve in the x_1 and x_3 axes, and the second variation pattern (s_2) controls the positioning of a point along the x_2 axis. The true tangent vectors $\mathbf{U}_n^{(1)} = [\cos(s_{n,1}) - s_{n,1}\sin(s_{n,1}),\ 0,\ s_{n,1}\cos(s_{n,1}) + \sin(s_{n,1})]$ and $\mathbf{U}_n^{(2)} = [0,\ 1,\ 0]$ are plotted in red at sampled points on the manifold.

We trained 30 ANN models with different randomly-initiated starting weights using normal backpropagation (no regularization) and using different combinations of

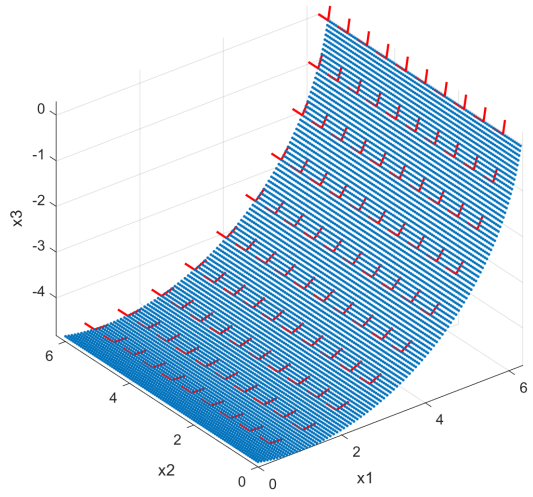

Figure 5.3: Scatter Plot of Simulated Data Points Prior to Adding Gaussian Noise With the True Tangent Vectors $\mathbf{U}_n^{(1)}$ and $\mathbf{U}_n^{(2)}$ Plotted in Red at Select Points

our regularization terms presented in Section 5.4. Specifically, we trained 30 models using each of the pairwise combinations of our proposed regularization penalties as well as a using all three of the terms. The same network structure depicted in Figure 5.1 with two bottleneck nodes and a single hidden layer on each side of the bottleneck layer was used.

Regularization parameter values of $\lambda_T = 0.01$, $\lambda_C = 0.01$, and $\lambda_K = 0.001$ were used for each model in which these regularization terms were present in the objective function. Each ANN model had $r_1 = r_3 = 75$ hidden nodes with sigmoid activation functions for the hidden layers on each side of the bottleneck layer while linear activation functions were used for the bottleneck nodes and output layer nodes. The models were therefore specified as 3-75-2-75-3 networks for the 3-dimensional inputs, 75 nodes in each hidden layer, and 2 nodes in the bottleneck layer. The objective function was minimized by (batch) stochastic gradient descent using batches of 150 data instances which were randomly selected without replacement in each epoch. These parameters were selected using the parameter search methodology described in Section 5.5.1.

A validation set of 100 data instances were withheld from the training data to monitor for overfitting. Each model was trained until 1000 epochs had elapsed since the minimum SSE of reconstruction on the validation set was achieved, at which point training was terminated and the weights achieving the minimum SSE were used for the final model. Each training epoch consisted of a single weight update using one batch.

Figure 5.4 provides boxplots detailing the performance of models with respect to the SSE of reconstructing the noiseless test data (left) as well as the VSS metric (right) across different training methods, where each boxplot shows the distribution of values across the 30 ANN models trained with the method. The five training methods depicted from left to right in each boxplot are normal backpropagation without our regularization terms (normal), $J_T(\mathbf{W})$ and $J_C(\mathbf{W})$ regularization (T+C), $J_C(\mathbf{W})$ and $J_K(\mathbf{W})$ regularization (C+K), $J_T(\mathbf{W})$ and $J_K(\mathbf{W})$ regularization (T+K), and regularization with all three of our proposed terms (T+C+K). The left plot of Figure 5.4 shows that each of our regularization methods resulted in similar median SSE when compared to normal backpropagation, but had slightly larger variance in the distribution of SSE values. The difference in quality of fit was not visually noticeable across the different methods when the reconstructed data points were plotted. The right plot of Figure 5.4 shows that the VSS is substantially lower on average for models trained with any combination of our proposed regularization terms than for models trained with normal backpropagation. The best performance with respect to consistently learning the true variation sources is provided when all three of our proposed regularization terms are used together.

An illustration of the manifold learned by an ANN model with and without our three regularization terms is provided in Figure 5.5. The left side of Figure 5.5 shows a surface plot of the reconstructed instances produced by a regular ANN model with

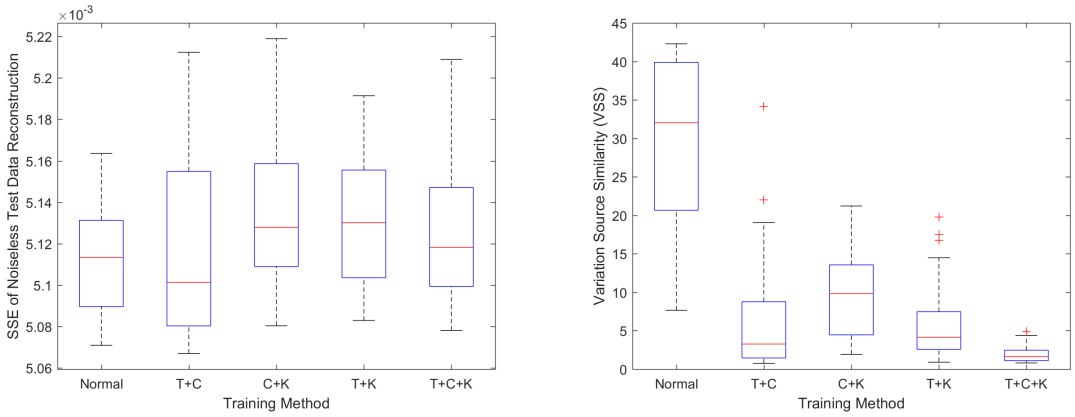

Figure 5.4: The Distribution of the SSE of Noiseless Test Data Reconstruction (Left) and VSS (Right) Across 30 Different Randomly Initialized 3-75-2-75-3 ANNs for Different Learning Methods

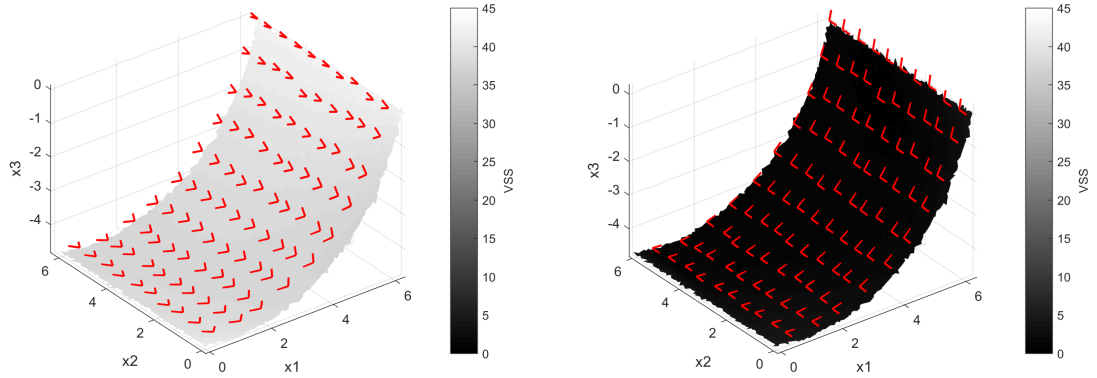

Figure 5.5: The Manifold Learned by a 3-75-2-75-3 ANN Model Trained Without Our Three Regularization Terms (Left) and With Our Three Regularization Terms (Right), Where the Tangent Vectors $\mathbf{T}_n^{(1)}$ and $\mathbf{T}_n^{(2)}$ Are Depicted as Red Vectors at Sampled Points on the Manifold

$VSS = 38.88$ while the right side of Figure 5.5 shows a surface plot of the reconstructed instances produced by an ANN model learned using our three regularization terms with $VSS = 1.55$. The tangent vectors $\mathbf{T}_n^{(1)}$ and $\mathbf{T}_n^{(2)}$ are depicted as red vectors at sampled points on the manifold. The plots were generated using the reconstructed data instances from each model; the surface color indicates the scale of the VSS across different locations on the manifold.

By comparing the learned manifolds in Figure 5.5 to the true simulated manifold

depicted in Figure 5.3, it is clear that the ANN model trained with our three regularization terms has learned the correct directions in the three-dimensional space characterized by the two variation sources. The learned tangent vectors $T_n^{(1)}$ and $T_n^{(2)}$ for this model are approximately equal to the negative of the true tangent vectors $U_n^{(1)}$ and $U_n^{(2)}$, therefore allowing the simulated variation sources to be accurately visualized by plotting the reconstructed instances as the bottleneck node values are independently changed. In contrast, the model trained without our regularization terms has learned features which represent a combination of the two true variation sources as evidenced by the orientation of the tangents in the left plot of Figure 5.5.

Manifold Coordinates Generated From the Normal Distribution

As described in Section 5.4, the $J_K(\mathbf{W})$ regularization term encourages the distribution of values for each of the bottleneck nodes to have the same kurtosis as the uniform distribution. This property is useful for our problem of identifying independent variation sources because it helps avoid complex interactions between the bottleneck node values by distributing the features uniformly over their range. In the case of the simulated siwss roll data depicted in Figure 5.3, the true variation sources $s_{n,1}$ and $s_{n,2}$ are uniformly distributed over their range because the data was generated by sampling at discrete intervals along the manifold. However, the distribution of $s_{n,1}$ and $s_{n,2}$ is unknown in practice and may not necessarily be uniform.

To evaluate the usefulness of $J_K(\mathbf{W})$ when the variation sources are not uniformly distributed, we simulated a different swiss roll data set where the variation source values were drawn from normal distributions with $s_{n,1} \sim N(\frac{7\pi}{4}, \frac{\pi}{12})$ and $s_{n,2} \sim N(3, 1)$. An example of the resulting $N = 10,000$ data points simulated from these normally-distributed $s_{n,1}$ and $s_{n,2}$ values is provided in Figure 5.6. Two sets of data were generated using this methodology and independent Gaussian noise was separately

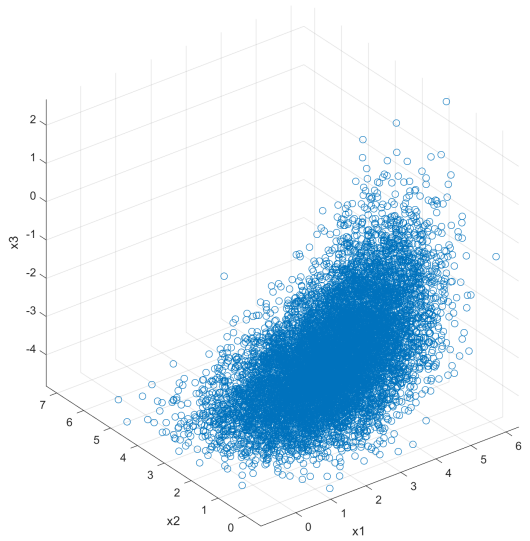

Figure 5.6: Scatter Plot of Normally-Distributed Simulated Data Points Prior to Adding Gaussian Noise

added to the two data sets using different random seeds.

We used one set of the normally distributed swiss roll data points to train 30 ANN models both with and without our three regularization terms and using the same parameter configuration described previously. The SSE and VSS was evaluated for each model using the other simulated data set which was not used during training. These models were trained using the same network structure and parameter configuration as the experiments described in the previous section. Specifically, the network structure depicted in Figure 5.1 with two bottleneck nodes and a single hidden layer on each side of the bottleneck layer was used, with $r_1 = r_3 = 75$ nodes having sigmoid activation functions in each hidden layer, 150 training instances randomly selected without replacement in each batch of stochastic gradient descent training, and regularization parameter values of $\lambda_T = 0.01$, $\lambda_C = 0.01$, and $\lambda_K = 0.001$ for models trained using our regularization method.

A boxplot showing the distribution of VSS values across the 30 models for each

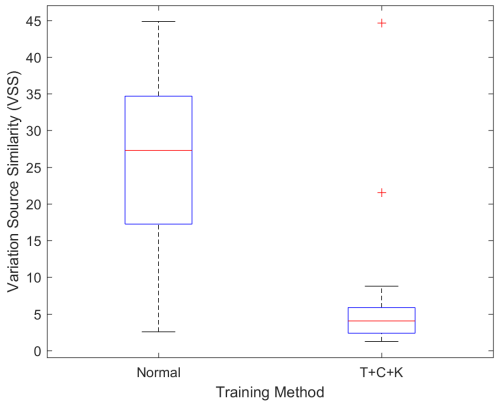

Figure 5.7: The Distribution of VSS Across 30 Different Randomly Initialized 3-75-2-75-3 ANNs for Different Learning Methods Evaluated Using the Normally Distributed Swiss Roll Test Set

method is provided in Figure 5.7. As before, Figure 5.7 shows that ANN models trained with our three regularization terms consistently outperform normal backpropagation with respect to learning the true variation sources. The median and standard deviation of SSE for normal backpropagation models were 0.0055 and 0.000043 (respectively), whereas the median and standard deviation of SSE for models trained with our three regularization terms were 0.0055 and 0.000079. Thus, the improved accuracy in learning the true variation sources came at only a small cost of higher standard deviation of SSE. More importantly, these results indicate that we do not need to assume the true variation sources $s_{n,1}$ and $s_{n,2}$ are uniformly distributed in order to use our proposed uniform excess kurtosis regularization term $J_K(\mathbf{W})$.

Greater Nonlinearity in Simulated Swiss Roll

To test the performance of our regularization method when faced with greater nonlinearity in the variation patterns, we simulated a larger version of the swiss roll data set and fit a set of models both with and without our regularization penalties. A scatter plot of the simulated data points used in these experiments is provided in Figure 5.8. The data points were simulated by generating equally-spaced values for

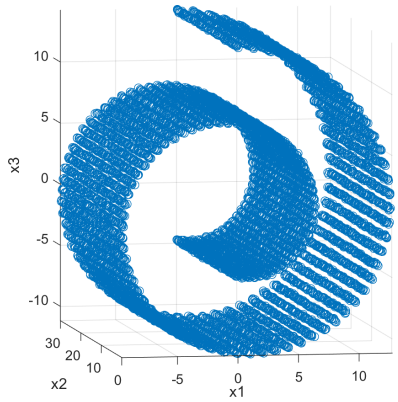

Figure 5.8: Scatter Plot of Simulated Swiss Roll With Greater Nonlinearity

the variation sources $s_1 \in [\frac{3\pi}{2}, \frac{9\pi}{2}]$ and $s_2 \in [0, 30]$. The set of $N = 10,000$ observed data instances were then generated using the functions $x_1 = s_1 \cos(s_1)$, $x_2 = s_2$, and $x_3 = s_1 \sin(s_1)$. Independent Gaussian noise with mean zero and standard deviation of 0.05 was also added to the three dimensions of each data instance. As with the previous experiments, separate training and testing data sets were simulated with the latter used only for final model evaluation.

A parameter search was performed to select the values of λ_T, λ_C, λ_K, the number of hidden nodes, and the number of training instances per batch using the same search methodology described previously. The best model configuration identified in the search had 200 nodes in each hidden layer, 200 training instances per batch, and values of $\lambda_T = 1$, $\lambda_C = 0.1$, and $\lambda_K = 1$. Therefore, the models are fully specified as 3-200-2-200-3 networks for the 3-dimensional input and output layers, 200 nodes in each hidden layer, and 2 nodes in the bottleneck layer. The same network structure depicted in Figure 5.1 with two bottleneck nodes and a single hidden layer on each side of the bottleneck layer was used for these models. Boxplots showing the distribution of VSS across 30 different randomly-initiated models trained with and without the three regularization terms are provided in Figure 5.9. Similar to the previous simulated data sets, models trained with our three regularization penalties achieved

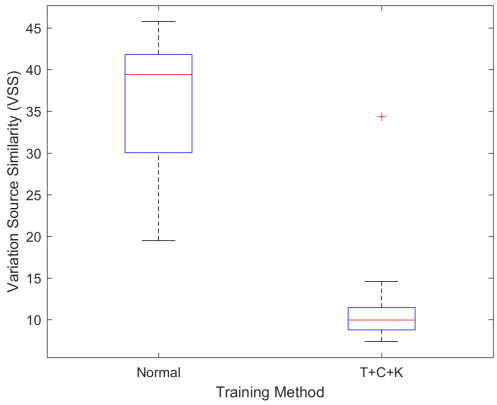

Figure 5.9: The Distribution of VSS Across 30 Different Randomly Initialized 3-200-2-200-3 ANNs for Different Learning Methods Evaluated Using the Swiss Roll Data Set With Greater Nonlinearity

substantially lower VSS values across different random weight initializations. They also achieved greater information preservation in the learned 2-dimensional representation, achieving a median of 0.1251 and a standard deviation of 0.4752 for the SSE of noiseless test data reconstruction. Models trained without our regularization penalties achieved a median SSE of 0.3679 with a standard deviation of 0.6753, suggesting that the constraints on learning imposed by our regularization method may also improve the efficiency of model training when the variation sources are more nonlinear.

Increasing Noise Levels

The previously-described experiments used variants of the simulated swiss roll with additive Gaussian noise having mean zero and a standard deviation of 0.05. To assess the sensitivity of our regularization method to different levels of noise in the observed data, we generated five different data sets consisting of the small portion of the siwss roll where the additive noise was drawn from a normal distribution with mean zero and differing standard deviations of 0.1, 0.25, 0.5, 1, and 2. Figure 5.10 provides

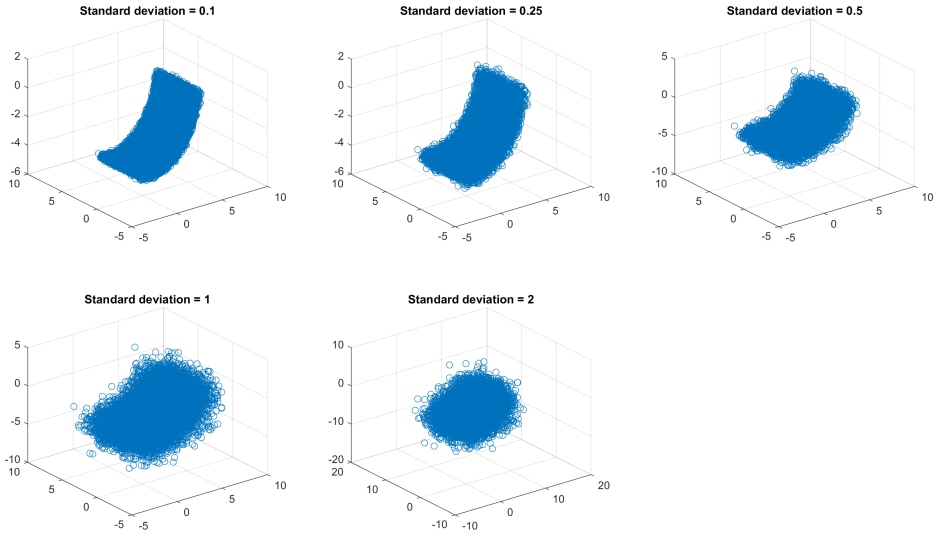

Figure 5.10: Scatter Plots of Swiss Roll Data Set Simulated With Increasing Levels of Additive Noise

scatter plots of the resulting simulated data points for each of the different noise levels.

We trained a set of 30 models on each of the five data sets both with and without our three regularization terms. Each model used the network structure depicted in Figure 5.1 having two bottleneck nodes and one hidden layer on each side of the bottleneck layer. The parameter configurations obtained from the previous set of experiments using the small portion of the swiss roll were used for each model, with 75 nodes in each hidden layer, 150 training instances per batch, and $\lambda_T = 0.01$, $\lambda_C = 0.01$, and $\lambda_K = 0.001$ used for the models trained with our regularization method. Boxplots depicting the distribution of the VSS for each training method and data set are provided in Figure 5.11 while the distribution of reconstruction SSE is depicted in Figure 5.12. In the figures, the noise level corresponding to each boxplot is indicated by the number in parentheses below the box. These results show that our regularization penalties continue to offer an improvement over normal backprop-

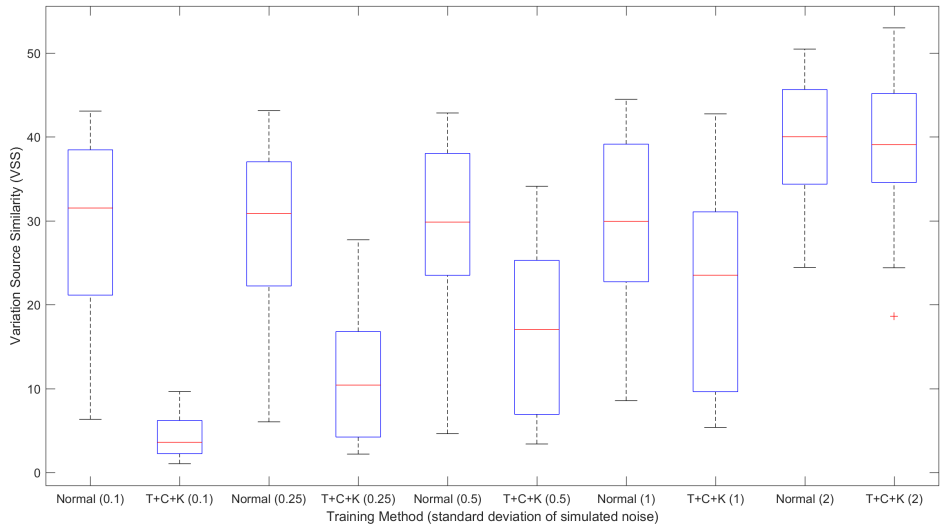

Figure 5.11: The Distribution of VSS Across 30 Different Randomly Initialized 3-75-2-75-3 ANNs for Different Learning Methods Evaluated Using the Swiss Roll Data Set With Five Different Additive Noise Levels

agation as the noise level increases, although higher levels of noise are also associated with higher VSS. Because VSS measures the angle between the ANN tangent vectors and the true tangent vectors across the data set, higher levels of VSS should be expected as the noise is increased due to a reduction in reconstruction accuracy and the changing direction of the tangent vectors as the reconstruction is located farther from the noiseless data instance on the manifold. Note that the difference in VSS performance between our regularization method and normal training only narrows considerably when the noise level is large enough to make the portion of the swiss roll nearly unidentifiable.

5.5.3 MNIST Handwritten Digit Images

To evaluate our proposed methodology on real data, we trained a set of 30 ANN models with and without our regularized learning method on images of the number zero from the MNIST handwritten digits data set (LeCun and Cortes, 1998). Because

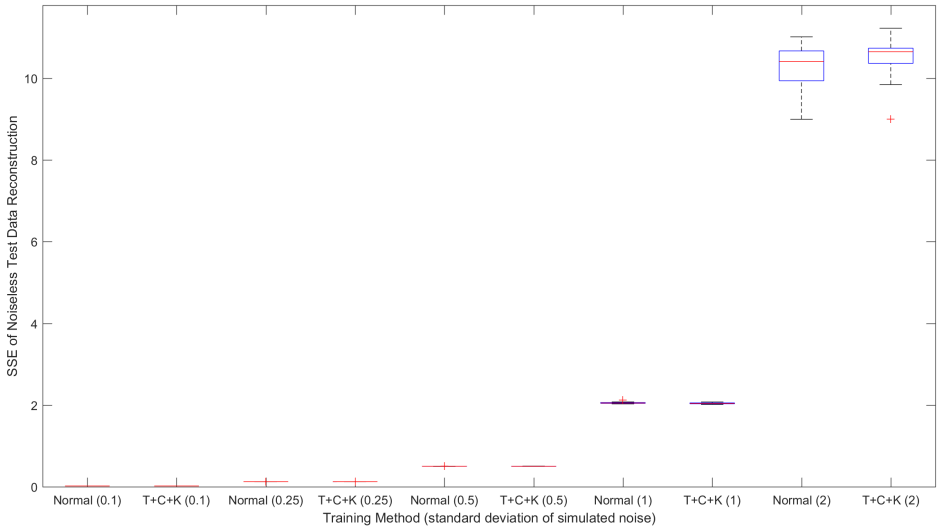

Figure 5.12: The Distribution of the SSE of Reconstruction Across 30 Different Randomly Initialized 3-75-2-75-3 ANNs for Different Learning Methods Evaluated Using the Swiss Roll Data Set With Five Different Additive Noise Levels

we do not have an equivalent of the noiseless data instances in our simulated swiss roll data, the fit of the models were evaluated using the SSE of reconstructing the original images. A VSS metric is not reported for any of the models fit to this data set because we lack information about the true variation sources governing this set of images.

The data set contained a total of 5923 images, of which 70% were used for training the models and 30% were withheld for testing. Each data instance had $M = 784$ dimensions and the ANN models were fit with $P = 2$ bottleneck nodes in order to learn two nonlinear features. As described previously in Section 5.5.1, the model parameters were selected from a parameter search using a portion of the training set for validation data. These parameters include $\lambda_T = 10$, $\lambda_C = 10$, and $\lambda_K = 1$ for the regularization terms, $r_1 = r_3 = 800$ nodes in each of the two hidden layers on either side of the bottleneck layer, and a batch size of 150 data instances randomly selected without replacement during each epoch for stochastic gradient descent training. The models

are therefore fully specified as 784-800-2-800-784 networks for the 784-dimensional input and output layers, 800 nodes in each hidden layer, and 2 nodes in the bottleneck layer. Each model was trained until 1000 epochs had elapsed since the minimum SSE of reconstruction on the validation set was achieved, at which point training was terminated and the weights achieving the minimum SSE were used for the final model. A training epoch consisted of a single weight update using one batch.

The set of models trained on this data using normal backpropagation had a median SSE of 27.2 with a standard deviation of 0.1045. The median and standard deviation of SSE was slightly larger for models trained using our three regularization terms at 27.5071 and 0.1610 (respectively). Because we do not know the true variation source values $\mathbf{s}_n$ or the nonlinear functions $\mathbf{f}(\mathbf{s}_n)$ which gave rise to this data, we cannot compute the VSS metric to numerically compare the two methods with respect to how well they have learned the true variation sources. However, we can visually compare the solutions produced by different models by plotting the 2-dimensional bottleneck node values and visualizing the reconstructed image at various locations on the learned manifold.

Figure 5.13 provides an illustration of the variation patterns learned by an ANN model trained using our three regularization terms. The scatter plot depicted in the left side of Figure 5.13 shows the $v_{n,1}$ and $v_{n,2}$ bottleneck node values evaluated across the entire data set (including both training and testing instances). The images reconstructed by this model were obtained at four sampled locations on the manifold, which are indicated by the numbers one through four in the scatter plot. The right side of Figure 5.13 depicts the corresponding reconstructed images where the location index is indicated by the white number in the top left corner of each grayscale image. The first bottleneck node $v_{n,1}$ controls the amount of interior space the digit has; movement from image one to two and three to four yields a widening of the digit

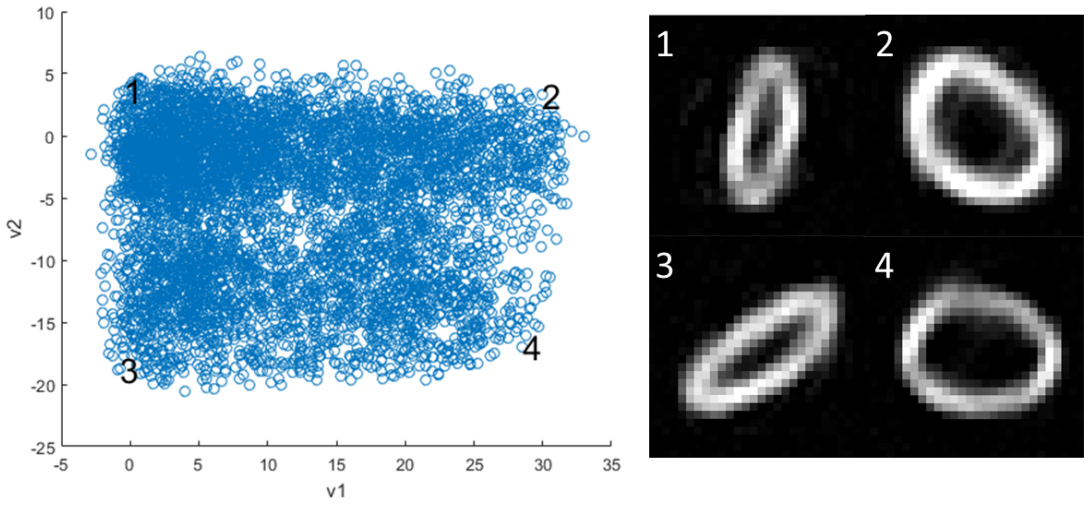

Figure 5.13: Bottleneck Node Coordinates (Left) and Reconstructed Images at Sampled Locations on Manifold (Right) Obtained From a 784-800-2-800-784 ANN Trained With Our Three Regularization Terms on the MNIST Data Set

while approximately maintaining its rotational orientation. The variation pattern learned by the second bottleneck node $v_{n,2}$ is a rotation of the orientation of each digit from left to right, which is evident by movement from image one to three and from image two to four. These variation patterns learned by the model are both interpretable and consistent across different locations on the manifold.

In contrast, Figure 5.14 depicts a solution obtained from an ANN trained using normal backpropagation which is much less interpretable. Changes in the $v_{n,1}$ bottleneck node appear to result in a widening of the digit when moving from image one to two, while movement from image three to four produces a thicker border for the digit. The second bottleneck node $v_{n,2}$ appears to have learned a combination of changing the interior space of the digit and producing some amount of rotation in orientation. Moreover, the patterns learned by each of the bottleneck nodes are much less consistent when they are visualized at different locations across the manifold. Figures 5.13 and 5.14 demonstrate how our three regularization terms yield nonlin-

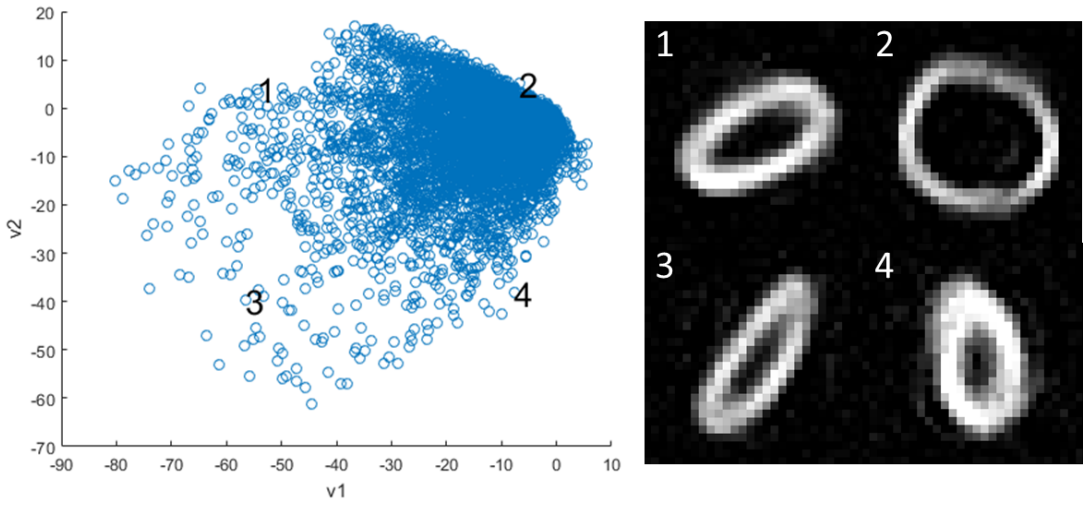

Figure 5.14: Bottleneck Node Coordinates (Left) and Reconstructed Images at Sampled Locations on Manifold (Right) Obtained From a 784-800-2-800-784 ANN Trained With Normal Backpropagation on the MNIST Data Set

ear features which are more interpretable and useful for understanding the principal sources of variation in the data.

5.6 Conclusion

We have presented a new regularization method for ANN models which encourages the learning of solutions where the true variation sources are correctly characterized by separate nonlinear components. Experimental results using simulated data demonstrates how our regularized learning method consistently yields ANN solutions which better characterize the true sources of variation and are therefore more interpretable for the purpose of analyzing high-dimensional data. Additionally, results obtained using a data set consisting of real images demonstrates how the nonlinear features extracted from models trained with our regularization terms are easier to interpret than those obtained form models trained without regularization.

The methodology presented in this work is applicable to a wide range of problem

domains in which understanding variation in high-dimensional data is important. Our results indicate that when confronted with the problem of learning low-dimensional representations of data, relying solely on error minimization is not sufficient for ensuring that knowledge of the variation sources can be extracted from the solution. Existing methods for nonlinear dimensionality reduction using ANNs are very efficient at learning a compact representation of high-dimensional data. However, they do not necessarily capture distinct features which can be used as a step towards interpreting what the model has learned. This work advances the current state of knowledge by introducing a method that provides efficient low-dimensional representations that are distinct and can be used to better understand the true variation sources giving rise to the observed data.

Although we have demonstrated three regularization terms in this work which improve the distinctness of features extracted from an ANN, we believe our general framework of using regularization to control internal properties of the model can be extended in future work. For example, it could be desirable depending on the problem at hand to enforce other constraints on ANN properties beyond those presented in this work. A natural question in utilizing additional regularization penalties is the practical limit for the number of constraints that can be effectively implemented during gradient descent training using regularization. In addition to these extensions of our regularization method, we also intend to explore alternative strategies for learning distinct nonlinear features by post-processing an ANN model trained with regular methods to make the features more distinct.

Chapter 6

CONCLUSION

The work presented in chapters 3, 4, and 5 contain several contributions to the existing body of research on discovering nonlinear variation patterns in high-dimensional data. In chapter 3, we demonstrated how recent advancements in pre-training of deep autoencoders can be applied to learning nonlinear variation patterns in multivariate spatial data. Compared to alternative approaches such as re-indexing and manifold learning, deep autoencoders produced a better model of the underlying noiseless variation patterns in terms of quality of fit. We also showed how deep autoencoders could be used to easily visualize the learned patterns by observing how the output of the network changes as the value at a single bottleneck node is changed over its observed range.

Chapter 4 presented a new method for learning autoencoder models which prevents the bottleneck nodes of the network from learning mixed combinations of the true nonlinear variation patterns present in the data. This is an important contribution enabling the use of autoencoders for learning distinct patterns when multiple underlying nonlinear effects are present. In addition to presenting the alternating autoencoder model for learning distinct patterns, we also presented the TVCS metric for quantifying the degree to which bottleneck nodes in the network have captured distinct and independent nonlinear variation patterns. This enables numeric comparisons of competing models and approaches with respect to their interpretability rather than simply relying on measures of the quality of fit.

In Chapter 5, we introduced a new regularization framework for training autoencoders in which the nonlinear features learned by the model represent distinct

variation sources. Motivated by the properties of linear dimensionality reduction solutions provided by PCA, this regularization method imposes several constraints on the model to encourage the learning of solutions which are more interpretable and distinct. These constraints include uncorrelated relationships between the bottleneck nodes of the model and orthogonal tangent vectors on the manifold learned by the model. Regularization methods are commonly used in the neural network literature for improving certain aspects of learning, such as penalizing complex models in order to improve generalization performance. Our regularization approach provides a fundamental direction that addresses the issue of distinct feature learning and represents a contribution to the existing literature on regularized learning for neural networks.

The results presented throughout this dissertation show that when confronted with the task of learning nonlinear variation sources, more advanced training techniques are needed than the typical method of minimizing the reconstruction error of the model. Existing work on autoencoders for dimensionality reduction has focused almost exclusively on optimizing the quality of fit of the model. While recent advances in this area of nonlinear dimensionality reduction have facilitated better information preservation in fewer dimensions, the ability to identify true variation sources governing the data has been sacrificed due to the tendency of autoencoders to learn non-distinct features. This work attempts to redirect the focus to identifying nonlinear low-dimensional representations that are both efficient in information preservation and useful for knowledge discovery through the learning of distinct features.

6.1 Future Work

The regularization framework we introduced in Chapter 5 included just three possible constraints on the properties of an autoencoder which have been shown to improve the distinctness of learned features. In practice, we expect there to be

other aspects of the model which could be useful to control through regularization depending on the problem context. Thus, we would like to explore alternative penalty terms that can improve autoencoders for the purpose of discovering distinct nonlinear variation sources.

Both Chapters 4 and 5 presented methodologies which alter the gradient descent training of an autoencoder. An alternative strategy for our task of learning distinct variation sources is to train an autoencoder using standard methods (e.g. minimizing the SSE of reconstruction) and then modifying the solution after training is complete to make the learned features more distinct. In future work, we would also like to explore this approach using methods motivated by the use of rotation matrices to improve the interpretability of linear dimensionality reduction solutions obtained from factor analysis.

Finally, both the alternating learning approach presented in Chapter 4 and the regularized learning approach introduced in Chapter 5 improved the distinctness of features learned by autoencoder models. The alternating method's approach of sequentially learning the variation sources in a hierarchical manner could be useful when combined with the regularization penalties presented in Chapter 5 to further improve upon the performance achieved by either method individually. Thus, we would like to explore a hybrid approach to sequentially learn distinct features in conjunction with enforcing constraints on the model's properties using regularization.

Alain, G. and Y. Bengio, "What regularized auto-encoders learn from the data-generating distribution", The Journal of Machine Learning Research **15**, 1, 3563–3593 (2014).

Apley, D. W. and H. Y. Lee, "Identifying spatial variation patterns in multivariate manufacturing processes: a blind separation approach", Technometrics **45**, 3, 220–234 (2003).

Bengio, Y., A. Courville and P. Vincent, "Representation learning: A review and new perspectives", Pattern Analysis and Machine Intelligence, IEEE Transactions on **35**, 8, 1798–1828 (2013).

Comon, P., "Independent component analysis, a new concept?", Signal processing **36**, 3, 287–314 (1994).

Dahl, G. E., T. N. Sainath and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout", in "Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on", pp. 8609–8613 (IEEE, 2013).

Duda, R. O., P. E. Hart and D. G. Stork, *Pattern classification* (John Wiley & Sons, 2012).

Erhan, D., Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent and S. Bengio, "Why does unsupervised pre-training help deep learning?", The Journal of Machine Learning Research **11**, 625–660 (2010).

Erhan, D., Y. Bengio, A. Courville and P. Vincent, "Visualizing higher-layer features of a deep network", Dept. IRO, Université de Montréal, Tech. Rep **4323** (2009a).

Erhan, D., P.-A. Manzagol, Y. Bengio, S. Bengio and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training", in "International Conference on artificial intelligence and statistics", pp. 153–160 (2009b).

Glorot, X. and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in "International conference on artificial intelligence and statistics", pp. 249–256 (2010).

Hinton, G., S. Osindero and Y.-W. Teh, "A fast learning algorithm for deep belief nets", Neural computation **18**, 7, 1527–1554 (2006).

Hinton, G. E., "Training products of experts by minimizing contrastive divergence", Neural computation **14**, 8, 1771–1800 (2002).

Hinton, G. E., "To recognize shapes, first learn to generate images", Progress in brain research **165**, 535–547 (2007).

Hinton, G. E. and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", Science **313**, 5786, 504–507 (2006).

Howard, P., D. Apley and G. Runger, "Identifying nonlinear variation patterns with deep autoencoders", IIE Transactions, submitted (2015).

Howard, P., D. Apley and G. Runger, "Distinct variation pattern discovery using alternating nonlinear principal component analysis", IEEE Transactions on Neural Networks and Learning Systems , submitted (2016a).

Howard, P. R., T. A. Reddy, G. C. Runger and S. Katipamula, "Automated data mining methods for identifying energy efficiency opportunities using whole-building electricity data", ASHRAE Transactions, to appear (2016b).

Im, J.-K., D. W. Apley and G. C. Runger, "Tangent hyperplane kernel principal component analysis for denoising", Neural Networks and Learning Systems, IEEE Transactions on **23**, 4, 644–656 (2012).

Jolliffe, I., *Principal component analysis* (Wiley Online Library, 2002).

Jolliffe, I., *Principal component analysis* (Wiley Online Library, 2005).

Jutten, C. and J. Karhunen, "Advances in nonlinear blind source separation", in "Proc. of the 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)", pp. 245–256 (Citeseer, 2003).

Kirby, M. and R. Miranda, "Circular nodes in neural networks", Neural Computation **8**, 2, 390–402 (1996).

Kramer, M. A., "Nonlinear principal component analysis using autoassociative neural networks", AIChE journal **37**, 2, 233–243 (1991).

LeCun, Y., Y. Bengio and G. Hinton, "Deep learning", Nature **521**, 7553, 436–444 (2015).

LeCun, Y. and C. Cortes, "The mnist database of handwritten digits", (1998).

Rifai, S., P. Vincent, X. Muller, X. Glorot and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction", in "Proceedings of the 28th international conference on machine learning (ICML-11)", pp. 833–840 (2011).

Rodgers, J. L., W. A. Nicewander and L. Toothaker, "Linearly independent, orthogonal, and uncorrelated variables", The American Statistician **38**, 2, 133–134 (1984).

Saegusa, R., H. Sakano and S. Hashimoto, "Nonlinear principal component analysis to preserve the order of principal components", Neurocomputing **61**, 57–70 (2004).

Sahu, A., D. W. Apley and G. C. Runger, "Feature selection for noisy variation patterns using kernel principal component analysis", Knowledge-Based Systems **72**, 37–47 (2014).

Schölkopf, B., A. Smola and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem", Neural computation **10**, 5, 1299–1319 (1998).

Scholz, M., M. Fraunholz and J. Selbig, "Nonlinear principal component analysis: neural network models and applications", in "Principal manifolds for data visualization and dimension reduction", pp. 44–67 (Springer, 2008).

Scholz, M. and R. Vigário, "Nonlinear pca: a new hierarchical approach.", in "ESANN", pp. 439–444 (2002).

Shan, X. and D. W. Apley, "Blind identification of manufacturing variation patterns by combining source separation criteria", Technometrics **50**, 3 (2008).

Shi, Z., D. W. Apley and G. C. Runger, "Identifying and visualizing part-to-part variation with optical dimensional metrology data", Journal of Quality Technology, submitted (2015).

Shi, Z., D. W. Apley and G. C. Runger, "Discovering the nature of variation in nonlinear profile data", Technometrics, to appear (2016).

Shinde, A., A. Sahu, D. Apley and G. Runger, "Preimages for variation patterns from kernel pca and bagging", IIE Transactions **46**, 5, 429–456 (2014).

Simard, P., Y. LeCun and J. S. Denker, "Efficient pattern recognition using a new transformation distance", in "Advances in neural information processing systems", pp. 50–58 (1993).

Vincent, P., H. Larochelle, Y. Bengio and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders", in "Proceedings of the 25th international conference on Machine learning", pp. 1096–1103 (ACM, 2008).