Local Models and Gaussian Mixture Models for Statistical Data Processing

Nandakishore Kambhatla
B.Tech, Institute of Technology, Benaras Hindu University, 1990

A dissertation submitted to the faculty of the
Oregon Graduate Institute of Science & Technology
in partial fulfillment of the
requirements for the degree
Doctor of Philosophy
in
Computer Science and Engineering

January 1996

The dissertation "Local Models and Gaussian Mixture Models for Statistical Data Processing" by Nandakishore Kambhatla has been examined and approved by the following Examination Committee:

Todd Leen
Associate Professor
Thesis Research Adviser

John Moody
Associate Professor

Mark Fanty
Research Assistant Professor

Andrew Fraser
Associate Professor
Portland State University

Misha Pavel
Associate Professor

Dedication

To my parents, Kambhatla Sriram and Kambhatla Kameswari

Acknowledgements

It seems incredible that the day has finally come, when I am writing this page! This work is a culmination of efforts by many people. I am indebted to all of them for making this work and this day possible.

First of all I would like to thank my advisor, Dr. Todd Leen, for his constant help and encouragement, and for the torrent of new ideas and suggestions which he directed at me. Any creativity that the reader finds in this thesis should be attribituted as much to him as to me. Thanks are also due to my other thesis committee members, Dr. Andrew Fraser, Dr. John Moody, Dr. Misha Pavel and Dr. Mark Fanty for reviewing this document so quickly and for their help and suggestions to improve this work.

I thank the members of the Center for Spoken Language Understanding at OGI and Dr. Gary Cottrell and Dr. David DeMers from UCSD for making their data and research available to me. I also thank Zoubin Ghahramani, MIT for many helpful comments and suggestions for the thesis.

I would like to thank all members of the "neural-local" community for the *high teas* and several stimulating conversations. A special thanks to Genevieve Orr and Steve Rehfuss, my officemates of many years for many invigorating discussions and their constant help and support, when things looked bleak. Finally, many thanks to my family members and many friends at OGI for their support and friendship. This document would not have been possible without them.

Contents

List of Tables

List of Figures

Abstract

Local Models and Gaussian Mixture Models for Statistical Data Processing

Nandakishore Kambhatla, Ph.D.
Oregon Graduate Institute of Science & Technology, 1996

Supervising Professor: Todd Leen

In this dissertation, we present local linear models for dimension reduction and Gaussian mixture models for classification and regression. When the data has different structure in different parts of the input space, fitting one global model can be slow and inaccurate. Simple learning models can quickly learn the structure of the data in small (local) regions. Thus, local learning techniques can offer us faster and more accurate model fitting. Gaussian mixture models form a *soft* local model of the data; data points belong to all "local" regions (Gaussians) at once with differing *degrees of membership*. Thus, mixture models blend together the different (local) models. We show that local linear dimension reduction approximates maximum likelihood signal extraction for a mixture of Gaussians signal-plus-noise model.

The thesis of this document is that "local learning models can perform efficient (fast and accurate) data processing". We propose local linear dimension reduction algorithms which partition the input space and build separate low dimensional coordinate systems in disjoint regions of the input space. We compare the local linear models with a global linear model (principal components analysis) and a global non-linear model (five layered

auto-associative neural networks). For speech and image data, the local linear models incur about half the error of the global models while training nearly an order of magnitude faster than the neural networks. Under certain conditions, the local linear models are related to a mixture of Gaussians data model.

Motivated by the relation between local linear dimension reduction and Gaussian mixture models, we present Gaussian mixture models for classification and regression and propose algorithms for regularizing them. Our results with speech phoneme classification and some benchmark regression tasks indicate that the mixture models perform comparably with a global model (neural networks).

To summarize, local models or Gaussian mixture models can be efficient tools for dimension reduction, exploratory data analysis, feature extraction, classification and regression.

Chapter 1

Introduction

In this dissertation, we consider local models for multi-variate data processing. We present local linear models for dimension reduction and Gaussian mixture models for classification and regression. We develop the relation between local linear dimension reduction and maximum likelihood signal extraction from a mixture of Gaussians signal-plus-noise model. Empirical results with speech and image data show that local models can perform fast and accurate data processing.

Multi-variate analysis is the statistical analysis of data that consists of sets of measurements. For example, for speech processing applications, we may model the frequency components of a segment of an utterance, and for image processing applications, we may model the pixel intensities of a digitized image. In both these cases, each pattern is characterized by a set of variables. Multi-variate analysis methods model the statistical dependence among these variables.

Unsupervised learning algorithms model the dependencies among the data variables without any external teacher or target function. In this dissertation, we present unsupervised algorithms for dimension reduction whose goal is to obtain low dimensional encodings of data vectors such that the mean squared error between the original vectors and their reconstructions from the low dimensional encodings is minimized. Dimension reduction algorithms are often used for extracting *features* for classification (Cottrell & Metcalfe 1991, Golomb, Lawrence & Sejnowski 1991, Leen, Rudnick & Hammerstrom 1990, Yang & Dumont 1991) and for image coding and compression (Cottrell, Munro & Zipser 1987, Cottrell 1988). For an n to m dimension reduction, these algorithms learn

encoding functions $f : \mathcal{R}^n \rightarrow \mathcal{R}^m$ and decoding functions $g : \mathcal{R}^m \rightarrow \mathcal{R}^n$, from sample data points.

Supervised learning algorithms model the dependencies between distinguished *input* and *output* variables. The goal is to predict the output vector for a given input vector. For classification, the output variables represent class labels, indicating membership to one of K categories or classes. Thus, classifiers learn a function $f : \mathcal{R}^n \rightarrow \{0,1\}^K$ based on sample data points. Regression algorithms learn a function $f : \mathcal{R}^n \rightarrow \mathcal{R}^m$ from n predictor variables (inputs) to m response variables (outputs), based on sample data points.

In this chapter, we first define global and local models and describe their relation to Gaussian mixture models. We then describe dimension reduction algorithms and briefly discuss local linear models for dimension reduction. In section 1.3, we discuss Gaussian mixture models for classification and regression. Finally, we sketch an outline of the rest of the thesis.

1.1 Global models, local models and Gaussian mixture models

Both unsupervised and supervised learning algorithms learn a function $f(\cdot)$, based on sample data points. Local learning algorithms employ a *divide-and-conquer* approach: they divide a complex problem into simpler problems and combine their solutions to yield a solution to the complex problem (Jordan & Jacobs 1994). A global model builds a single function for the whole data space, while a local model builds separate functions in different regions of the data space. For instance, consider learning a function $f : \mathcal{R}^n \rightarrow \mathcal{R}^m$. A local model partitions $\mathcal{R}^n$ into Q disjoint regions $\{R_1, \ldots, R_Q\}$ and learns separate *local* functions $f_i : R_i \rightarrow \mathcal{R}^m$, for $i = 1, \ldots, Q$. Thus the model is

$$f(x) = \begin{cases} f_1(x) & \text{if } x \in R_1. \\ \vdots & \\ f_Q(x) & \text{if } x \in R_Q. \end{cases}$$

A feedforward neural network (Rumelhart, Hinton & Williams 1986, Hertz, Krogh & Palmer 1991) builds a global model of the data. Examples of a local model include the *classification and regression trees* (CART) algorithm (Breiman, Friedman, Olshen & Stone 1984) and Friedman's (1991) *multivariate adaptive regression splines* (MARS) algorithm. These algorithms divide the input space into a *nested* set of regions and fit simple functions (e.g a constant function or a low order polynomial function) within these regions. Leen and I propose local linear models for dimension reduction (Kambhatla and Leen 1993,1994). Hinton *et al* (Hinton, Revon & Dayan 1995) use local linear models for handwritten digit recognition.

Global models can be inefficient (slow to learn and inaccurate) whenever the structure of the data is different in different parts of the input space, i.e whenever the function $f(\cdot)$ is highly complex. For instance, if the n dimensional data lies near an k dimensional curved manifold[1], a global model can potentially incur a large error since, an inaccurate fit in one region of the input space effects the fit in all of the input space (see section 2.2.4 for an example of this). Simple models (e.g a constant function or a linear function) can quickly learn the structure of the data in local regions of the input space. Local models like CART and MARS often have convergence times orders of magnitude faster than neural network algorithms (Jordan & Jacobs 1994).

In terms of the *bias/variance* tradeoff (Gemen, Bienenstock & Doursat 1992), in general, a local model will have a higher variance than a global model having equal complexity as each local model, since the local parameters are estimated based on a smaller number of data points (subsets of the original training set). We can lower the variance of the local models by using "*soft partitions*" of the data[2], where data points are allowed to simultaneously belong to multiple regions with differing *degrees of membership* (Basford & McLachlan 1985, McLachlan & Basford 1988, Nowlan 1991, Jordan & Jacobs

[1]This means that the data surface is diffeomorphic to $\mathcal{R}^k$ in the neighborhood of any data point on the surface.

[2]Soft partitions are referred to as *probabilistic clustering* in the statistics community, e.g see (Basford & McLachlan 1985, McLachlan & Basford 1988).

1994). A *soft local* model of the data blends together different (soft local) functions.

$$f(x) = \sum_{i=1}^{Q} h_i(x) f_i(x) \quad , \tag{1.1}$$

where $f_i(\cdot)$ are the soft local functions, and $h_i : \mathcal{R}^n \rightarrow \mathcal{R}$ represent the probability that x belongs to the support of $f_i(\cdot)$. Thus, a soft local model computes a weighted sum of (local) functions. Jacobs, Jordan *et al* propose soft local models for supervised learning as a "mixture of experts" architecture (Jacobs, Jordan, Nowlan & Hinton 1991, Jordan & Jacobs 1994). Bregler and Omohundro use soft local linear models to learn non-linear constraint surfaces from the data (1994) and for "manifold learning" (1995). Marroquin (1995) discusses soft local models and local models for classification and regression.

A model of the probability density function of an n dimensional random vector x as a mixture density of Q multi-variate Gaussians defines a soft local model of the data. Let

$$\begin{aligned} p(x) &= \sum_{i=1}^{Q} \alpha_i \, p_i(x) \\ &= \sum_{k=1}^{Q} \frac{\alpha_i}{(2\pi)^{n/2}\sqrt{|\Sigma_i|}} \exp\left[-\frac{1}{2}(x-\mu_i)^T \, {\Sigma_i}^{-1} \, (x-\mu_i)\right] \end{aligned} \tag{1.2}$$

where $p_i(x)$ are the component Gaussian densities, and μ_i and Σ_i are the means and covariance matrices of the i^{th} component Gaussian. Note the similarity between (1.2) and (1.1). Each component Gaussian $p_i(\cdot)$ is a *soft local* function of the inputs and the mixture density $p(\cdot)$ is a weighted sum of the soft local functions.

We can obtain a hard partition from the Gaussian mixture model by assuming that for any given x, the mixture density $p(x)$ is dominated by only one of the component Gaussians. Mixture models have been extensively used as a hard clustering technique in the statistics community (e.g see (Lazarsfeld & Henry 1968, Ganesalingam & McLachlan 1979, Aitkin, Anderson & Hinde 1981, Basford & McLachlan 1985)). McLachlan and Basford (1988) note that one can obtain a hard partition from a mixture model by allocating each vector x to the component to which x has the highest posterior probability of belonging. Nowlan (1991) used this "*winner-take-all*" assumption to show the relation

between Gaussian mixture models and hard clustering algorithms like vector quantization (VQ; (Gersho & Gray 1992)) which build a local data model. The relation between VQ clustering and a mixture of Gaussians model was earlier suggested by Duda and Hart (1973). In appendix B, we summarize a derivation of this relation.

In this dissertation, we present local linear models for dimension reduction and Gaussian mixture models for classification and regression. We show that local linear dimension reduction algorithms approximate a Gaussian mixture model. Gaussian mixture models for classification define a soft local model of the data and Gaussian mixture models for regression define a soft local linear regression function. In the next section, we describe the *curse of dimensionality* and discuss local linear dimension reduction algorithms. We then describe Gaussian mixture models for classification and regression.

1.2 The "curse of dimensionality"

The "curse of dimensionality" is a phrase coined by Bellman (1961) to refer to problems associated with high dimensional data. The basic problem is that data is very sparse in high dimensions. As Breiman *et al* (1984) note in their book, 10 points on the unit interval are a lot closer together than 10 points in a unit 10-dimensional hypercube. If we build a histogram with 10 intervals in each dimension, we will have 10^n cells for n dimensional data. For large n, a very large number of data points would be required to obtain a reasonable histogram.

A related problem is the potential over-parametrization of data processing algorithms, when modelling high dimensional data. For most commonly used density estimation procedures, the number of parameters grows much faster than $O(n)$ with increasing n (input dimensionality). For example, a multi-variate Gaussian density has $O(n^2)$ parameters, a multinomial density has $O(2^n)$ parameters, and a mixture density of Q Gaussians has $O(Q*n^2)$ parameters. Unless we regularize these models, we require more complex models (with many parameters) to model the statistical dependencies among the data variables for high dimensional data.

If the number of parameters of a model is large relative to the number of sample data points used to train the model, the model can *over-fit* to the training data. Thus, the model may learn spurious dependencies that do not exist in the population, resulting in poor generalization ability. In terms of the bias/variance tradeoff (Gemen *et al.* 1992), over-parametrized models can have a high model variance, which means that the trained parameter values vary greatly with the particular training set used. Complex models with a large number of parameters can also require a long time to train and a large storage space.

In contrast, low dimensional data is easier to visualize and interpret, and is easier to model, since small sample data sets are sufficient to learn the data dependencies of the population. Moreover, the models are less compute and memory intensive. In the next section, we describe algorithms for reducing the dimensionality of data.

1.2.1 Dimension reduction algorithms

The objective of dimension reduction algorithms is to obtain a parsimonious description of the structure underlying a set of multi-variate data. The goal is to obtain *compact* encodings which optimize a criterion for "accuracy" of representation. In general, the criteria will depend upon the specific application.

For instance, for classification tasks, we want to extract features from the data which are most effective for separating the classes. In this case, "accuracy" refers to the "effectiveness for separating classes". Linear discriminant analysis (Fukunaga 1972) finds the optimal linear transformation of the data vectors to extract (linear) features based on their effectiveness for separating classes.

Factor analysis (Harman 1976) is a dimension reduction technique, which aims to obtain low dimensional encodings which maximally account for the correlations between the original data variables. Multi-dimensional scaling (Kruskal 1964) is a technique for extracting the underlying criteria or dimensions that people use in perceptually measuring (dis)similarities between objects (data points).

We measure the "accuracy of representation" by the mean squared error induced in

reconstructing the original data vectors from the low dimensional encodings. This accuracy criterion is often used for extracting *features* for classification (Cottrell & Metcalfe 1991, Golomb *et al.* 1991, Leen *et al.* 1990, Yang & Dumont 1991) and for image coding and compression (Cottrell *et al.* 1987, Cottrell 1988). The goal is to obtain compact representations of the data, from which we can best recover the original data (in the mean squared sense). Let $f : \mathcal{R}^n \rightarrow \mathcal{R}^m$ define an encoding function from a vector $x \in \mathcal{R}^n$ to a vector $z = f(x) \in \mathcal{R}^m$, where $m < n$. Let $g : \mathcal{R}^m \rightarrow \mathcal{R}^n$ define a decoding function from z to $\hat{x} = g(f(x)) \in \mathcal{R}^n$, a reconstruction of x. The mean squared error in reconstructing the original data is

$$\mathcal{E} = E\left[\| x - g(f(x)) \|^2 \right] \quad , \tag{1.3}$$

where $E[\cdot]$ denotes an expectation with respect to the random vector x.

Principal component analysis (Hotelling 1933, Oja 1983) (PCA) is a dimension reduction algorithm which obtains the least error (1.3) among all techniques with linear encoding and decoding functions ($f(\cdot)$ and $g(\cdot)$). PCA builds a *global linear* model of the data: an m dimensional hyperplane spanned by the leading eigenvectors[3] of the data covariance matrix. PCA incurs a high error whenever the data has non-linear dependencies not eliminated by removing correlations.

Five layered auto-associative neural networks (FLNs) (Kramer 1991, Oja 1991) are capable of building *non-linear* encoding $f(\cdot)$ and decoding $g(\cdot)$ functions, by capturing any non-linear dependencies among the data variables. These networks have at least one layer with a smaller number of nodes (m nodes) than the number of inputs (n). The networks are trained to learn an identity mapping. The activations of the bottleneck layer nodes form the low dimensional representation. FLNs can form a *global non-linear* model of the data: a smooth low dimensional (possibly curved) non-linear surface that approximates the spread of the data. A FLN approximates a data point $x \in \mathcal{R}^n$ by projecting it onto the learned manifold.

[3]The leading m eigenvectors of a positive definite matrix are the eigenvectors which correspond to the largest m eigenvalues.

Training big five layered networks can be time consuming and inefficient. In general, a global model can be efficient (inaccurate and hard to fit) when the data lies near complex curved manifolds. A subset of $\mathcal{R}^n$, X is an m dimensional manifold if it is locally diffeomorphic to $\mathcal{R}^m$. In other words, X is an m dimensional manifold, if near any point $x \in X$, there exists a neighborhood, from which we can build a smooth, one-to-one and onto mapping to $\mathcal{R}^m$ whose inverse is also smooth. A global model for dimension reduction applies a single encoding and decoding function $f(\cdot)$ to all regions of the input space. When the data has different structure in different regions of the input space, this can be very inefficient. Building simple linear models in *local regions* of the input space can be faster and more accurate.

We propose the following algorithm:

1. Partition $\mathcal{R}^n$ into Q disjoint regions $\{R_1, \ldots, R_Q\}$.

2. Form separate *linear* encoding and decoding functions $f_i(\cdot) : R_i \to \mathcal{R}^m$ and $g_i(\cdot) : \mathcal{R}^m \to \mathcal{R}^n$ in each R_i, $i = 1, \ldots, Q$, using the subset of the training data mapped onto R_i in the first step.

3. To reduce dimension of $x \in \mathcal{R}^n$, use the encoding function

$$f(x) = \begin{cases} f_1(x) & \text{if } x \in R_1. \\ \vdots & \\ f_Q(x) & \text{if } x \in R_Q. \end{cases} \tag{1.4}$$

The encoding is given by $< w, z >$, where w is the index of the local region and $z = f_w(x) \in \mathcal{R}^m$ is the local m dimensional coordinate vector. The decoding function is

$$g(z, w) = g_w(z) \ .$$

We use a *vector quantizer* (VQ; a piece-wise constant modelling technique which partitions the input space into a disjoint regions) to partition the input space in step 1 above and an m dimensional PCA to build the local encoding and decoding functions. We refer to the above procedure as VQPCA.

Algorithms based on PCA in local regions of the input space have been used before. Fukunaga (1971) and Broomhead (1991) use PCA in local regions to estimate the intrinsic dimension of the data. We propose the VQPCA model in (Kambhatla & Leen 1993, Kambhatla & Leen 1994). Bregler and Omohundro (1994, 1995) use PCA in local regions to learn constraint surfaces of the data and to interpolate between image sequences. Hinton *et al* (1995) use an algorithm based on local PCA for handwritten character recognition.

In summary, we propose local linear models for dimension reduction which build several low dimensional coordinate systems in disjoint regions of the input space. We will show that this method is more efficient (more accurate and faster to train) than learning a global non-linear surface which approximates the data.

In the first part of this thesis, we present PCA, FLNs, and several implementations of VQPCA which differ in the method used to partition the input space. We derive the relation of VQPCA to a mixture of Gaussians data model and present empirical results comparing PCA, FLNs and VQPCA. In the second part of the thesis, we present Gaussian mixture models for supervised learning algorithms and methods for regularizing them. In the next section, we briefly describe these algorithms.

1.3 Gaussian mixture models for supervised learning

References to the use of statistical mixture models as a modelling technique date back to the late 19th century papers by Newcomb (1886) and Pearson (1894). Mixture models are extensively used in applications where data can be viewed as arising from several populations mixed in varying proportions. Gaussian mixture models have been used for the identification of outliers (Aitkin & Wilson 1980), and for the investigation of robustness of certain statistics to departures from normality (Srivastava & Lee 1984). Gaussian mixture models are widely used to tackle the "missing data problem" where one or more of the data variables may be unavailable (Dempster, Laird & Rubin 1977, Ghahramani

1994, Ghahramani & Jordan 1994*b*). Mixture models have been used as a clustering technique (Lazarsfeld & Henry 1968, Ganesalingam & McLachlan 1979, Basford & McLachlan 1985), for supervised learning in the "mixture of experts" architecture (Jacobs *et al.* 1991, Jordan & Jacobs 1994), for classification using Bayes classifiers (Ghahramani & Jordan 1994*a*, Ghahramani & Jordan 1994*b*, Ormoneit & Tresp 1995, Hinton *et al.* 1995), and for regression (Ghahramani & Jordan 1994*a*, Ghahramani & Jordan 1994*b*, Leen 1995). For a much more extensive discussion on mixture models and their myriad applications, see the books by Everitt and Hand (1981), Titterington *et al* (1985) and McLachlan and Basford (1988). In particular, the text by Titterington *et al* contains a comparative table of 90 applications of mixture models to real world problems. In this section, we describe Gaussian mixture models for classification and regression.

1.3.1 Gaussian mixture models for classification

A classifier assigns vectors from $\mathcal{R}^n$ (n dimensional feature space) to one of K classes or categories, partitioning the feature space into K disjoint regions. We model the input variables for each class *separately* using a mixture of Gaussians (1.2) and build Bayes classifiers (called Gaussian mixture Bayes (GMB) classifiers).

Priebe and Marchette (1991) describe a classification algorithm which uses a recursive technique derived from Gaussian mixture models for each class. They recursively fit mixture models for each class, where the number of components is allowed to grow with the data. Ghahramani and Jordan (1994*a*, 1994*b*) and Ormoneit and Tresp (1995) have previously discussed GMB classifiers. Heck and Chou (1994) use a GMB model to classify machine failure modes. They classify a signal as containing metallic or non-metallic transients based on a time frequency representation of the signal using the wavelet transform. Marroquin (1995) describes "local Gaussian classifiers" where he models each class conditional density as a mixture of Gaussians, and uses Bayes discriminant functions to classify input vectors.

In this dissertation, we study GMB classifiers, show their relation to clustering based classification algorithms and explore different ways of regularizing them. We train the

mixture models using the *Expectation Maximization* (EM) algorithm (Dempster *et al.* 1977) (see appendix C); an iterative algorithm for generating maximum likelihood parameter estimates of mixture models. We derive winner-take-all approximations to GMB classifiers and show that clustering based classifiers like the learning vector quantization algorithm (Kohonen 1988) (LVQ; see appendix D) approximate a GMB model.

GMB classifiers suffer from the curse of dimensionality. A GMB classifier with a mixture of Q Gaussians for each of K classes and for n dimensional data has $O(K*Q*n^2)$ parameters. We explore the following ways of regularizing GMB classifiers:

- A commonly used technique is to assume that all covariance matrices are diagonal (Nowlan 1991, Ahmad & Tresp 1993) or spherically symmetric (Stensmo & Sejnowski 1995, Nowlan 1991). This greatly reduces the number of parameters. However, this assumption is often invalid. For instance, when classifying images using pixel intensities, the intensities of neighboring pixels are very highly correlated and hence the covariance matrix is far from diagonal or spherically symmetric.

- Following (Ghahramani & Jordan 1994*a*, Ormoneit & Tresp 1995), we add a constant diagonal matrix $\epsilon I_{n\times n}$ to each covariance matrix in each iteration of the EM algorithm for training the mixture model for each class. This adds bias to the model by imposing an artificial lower bound on the volume elements (determinants of covariance matrices) of each Gaussian, and can decrease the model variance.

- We propose a new method of regularizing GMB classifiers, which prunes those eigen-directions of each covariance matrix which induce the least (empirically measured) classification error when pruned.

We present experimental results comparing GMB classifiers, regularized GMB classifiers and a feedforward neural network for two speech phoneme classification tasks. Our results indicate that GMB classifiers perform comparably to neural networks and the performance is improved by regularization. The diagonal and spherically symmetric assumptions resulted in the worst performance among the schemes sketched above.

1.3.2 Gaussian mixture models for regression

The goal of regression analysis is to predict values of m response variables y, given observed values of n predictor variables x. We model the joint density of the inputs and outputs as a mixture of Q Gaussians (1.2), and derive the regression function $E[y \mid x]$ as in (Ghahramani & Jordan 1994*a*, Ghahramani & Jordan 1994*b*). The regression is a weighted sum of linear models, where the weights are the probabilities of membership to the component Gaussians of the mixture model. Thus, a Gaussian mixture model for regression defines a *soft local linear* model of the data (see section 1.1).

We present the Gaussian mixture regression (GMR) algorithm which trains a mixture of Gaussians model (using the EM algorithm) for the joint density of the predictor and response variables, and uses the estimated regression function $E[y \mid x]$ to predict the response variables. The mixture of Gaussians model can suffer from the curse of dimensionality since it has $O(Q * (n + m)^2)$ parameters (the joint density is $n + m$ dimensional). We propose two new ways (Leen 1995) of regularizing the regression function:

- local ridge regression, where we regularize each of the local predictor functions using ridge regression (Draper & Smith 1981) and
- principal components pruning (PC pruning), where we prune those directions of each local prediction matrix which induce the least additional error when pruned (similar to (Levin, Leen & Moody 1994)).

We present experimental results comparing GMR, regularized GMR algorithms, and a feedforward neural network for some benchmark regression tasks. Our results suggest that the GMR algorithms performed comparably to the neural networks, and regularization was useful for high dimensional data, when the sample size was small.

In this section, we described Gaussian mixture models for classification and regression and algorithms for regularizing these models. In the next section, we outline this document.

1.4 Outline of thesis

In the first part of the thesis, we discuss dimension reduction algorithms. In chapters 2 through 5, we describe global and local models for dimension reduction, a probabilistic framework for the local linear models, and experimental results comparing the different models for speech and image dimension reduction. In chapter 2, we discuss two global models for dimension reduction. We describe principal components analysis (PCA), a global linear model, and five layered auto-associative neural networks (FLNs), a global non-linear model.

In chapter 3, we present our local linear models for dimension reduction (VQPCA model) which partition the input space using a vector quantizer (VQ), and build separate PCA models in the disjoint regions. We present several algorithms for implementing VQPCA, which differ in the method used to obtain the VQ partition. We propose a new distortion measure for VQ, called "reconstruction distance" which measures the squared distance of a data point to the m dimensional (local) PCA hyperplane. Clustering using the reconstruction distance directly minimizes the squared reconstruction error (1.3) for VQPCA. We describe efficient algorithms for partitioning the input space using a hierarchical multi-stage or a tree structured VQ, which trade-off a loss in accuracy for a reduction in complexity.

In chapter 4, we describe a probabilistic framework for VQPCA based on mixture of Gaussians signal-plus-noise model. We show that, under certain conditions, the VQPCA encoding is related to the maximum likelihood signal estimate for a probabilistic signal-plus-noise model, where

- the signal density is a mixture of Gaussians, each with with $n - m$ eigen-directions with negligibly small eigenvalues,
- the noise density is a spherically symmetric Gaussian.

In chapter 5, we present experimental results comparing the different models discussed above for speech and image dimension reduction, speech feature extraction and

speech and image compression. Our results with dimension reduction experiments suggest that local linear models (VQPCA) obtain nearly half the error obtained by global models (FLNs or PCA) and train nearly an order of magnitude faster than a global non-linear model (FLNs). We show that dimension reduction algorithms can be used for speech feature extraction, by learning mappings from low dimensional representations of speech spectral coefficients to *formant frequencies* (resonant frequencies of the vocal tract). We compare VQ and VQPCA for lossy compression of speech and image data. We find that, in general, a VQ performs the most accurate compression for a given bit-rate.

In chapters 6 and 7, we present regularized Gaussian mixture models for classification and regression. In chapter 6, we present Gaussian mixture models for classification and new ways of regularizing the models. We present Gaussian mixture Bayes (GMB) classifiers which use a mixture of Gaussians model for each class conditional density. We show the relation between GMB classifiers and hard clustering based classifiers like the learning vector quantization algorithm[4] (LVQ; see (Kohonen 1988) and appendix D). We discuss different ways of regularizing GMB classifiers and propose a new regularization scheme, which prunes those eigen-directions of each discriminant function, which induce the least (empirically measured) classification error when removed. This reduces the model variance, and induces additional bias. We compare the performance of different mixture models and a feedforward neural network for two speech phoneme classification tasks. Our results suggest that GMB classifiers perform comparably to neural networks and regularizing the mixture models is necessary to prevent singular covariance matrices and over-parametrization.

In chapter 7, we present Gaussian mixture models for regression and propose new ways for regularizing them. We model the joint density of the inputs and outputs as a mixture of Gaussians, and derive the regression function $E[y \mid x]$; a local linear function of the inputs. We propose two ways of regularizing the regression function: local

[4]LVQ is a clustering based classifier which adapts the placement of *reference vectors* based on a set of class labelled data points.

ridge regression, where we regularize each of the local predictor functions using ridge regression and principal components pruning (PC pruning), where we prune those directions of each prediction matrix which induce the least additional error when pruned. We present results of experiments predicting the Boston housing prices, the average monthly sunspots count and a chaotic time series. Our results suggest that regularizing the local linear predictors was useful for high dimensional data, when the sample size was small. Finally, in chapter 8, we give a summary of the whole thesis, present our conclusions and suggest directions for future work.

Chapter 2

Global models for dimension reduction

In this chapter, we describe two algorithms for dimension reduction which build a global model of the data: principal component analysis and five layered auto-associative neural networks.

The objective of dimension reduction is to obtain a compact representation of the data for combating the curse of dimensionality (see section 1.1) while retaining sufficient accuracy of representation. We define dimension reduction as the transformation of an n dimensional vector x ($x \in \mathcal{R}^n$) to an m dimensional vector z ($z \in \mathcal{R}^m$), where $m < n$. Implicit in this definition is an encoding function $f : \mathcal{R}^n \rightarrow \mathcal{R}^m$ from the inputs x to the low dimensional encodings z. The idea of "accuracy of representation" can be made precise by defining a decoding function $g : \mathcal{R}^m \rightarrow \mathcal{R}^n$ from the encoding z to the input space. When m is specified, the goal is to minimize some suitably defined error measure between x and $g(f(x))$. We use the mean squared error defined as

$$\mathcal{E} = E\left[\| x - g(f(x)) \|^2 \right] \quad , \tag{2.1}$$

to characterize the accuracy of representation.

Different dimension reduction algorithms differ in the mappings $f(\cdot)$ and $g(\cdot)$ that they build. Global algorithms define a single encoding function $f(\cdot)$ and a decoding function $g(\cdot)$ for all data points. Local algorithms define different encoding functions in

different regions of the input space. Thus, a local algorithm might define $f(\cdot)$ as

$$f(x) = \begin{cases} f_1(x) & \text{if } x \in S_1. \\ \vdots & \\ f_Q(x) & \text{if } x \in S_Q. \end{cases} \quad , \tag{2.2}$$

where the sets $S_1, \ldots, S_Q$ define a partition of $\mathcal{R}^n$ and $f_1(\cdot), \ldots, f_Q(\cdot)$ are local encoding functions.

In this chapter, we describe two global algorithms for dimension reduction. We first present principal components analysis (PCA), a global linear algorithm. We describe PCA, its optimality properties and various implementations of it. We then describe five layered auto-associative networks which are capable of forming a global non-linear model of the data.

2.1 Principal components analysis (PCA)

Principal components analysis (PCA) is a classical technique for dimension reduction. PCA builds a *global linear* model of the data; an m dimensional hyperplane spanned by the leading m eigenvectors of the data covariance matrix. PCA is the *optimal* linear technique, since, among all linear techniques, it obtains the minimum expected squared distance between an input vector and its reconstruction from an m dimensional encoding.

PCA was first proposed by Hotelling (1933) for dimension reduction. Anderson (1958) and Morrison (1976) use PCA to reduce the number of variables by eliminating linear combinations with small variance. Oja (1983) discusses PCA and related techniques. A closely related orthogonal expansion to PCA is the Karhunen-Loeve (K-L) expansion (Watanabe 1965) which was originally conceived in the framework of continuous second-order stochastic processes. When restricted to a finite dimensional case and truncated after a few terms, the K-L expansion is equivalent to a PCA expansion.

PCA is used for feature extraction and data compression (Devijver & Kittler 1982), image compression (Jain 1989) and characterization of signals in signal processing (Wax & Kailath 1985). Fukunaga and Olsen (1971) and Hediger *et al* (Hediger, Passamante

& Farrell 1990) use PCA in local regions of the input space to determine the intrinsic dimensionality of a data set. PCA is widely used as a preprocessor for pattern recognition applications to reduce the input dimension before building classifiers. PCA generated encodings of images have been used to classify faces, emotions and gender (Cottrell & Metcalfe 1991), and the sexes of humans (Golomb *et al.* 1991). Leen *et al* (1990) show that using the principal components of speech data for classification reduces the training time significantly without any adverse effect on the accuracy. Yang and Dumont (1991) show that the use of PCA as a front end for the classification of acoustic emission signals reduces the training time for the classifier, while preserving the classification accuracy. PCA has also been used for the coding of gray-scale images (Cottrell *et al.* 1987) and image compression (Cottrell 1988). Several artificial neural networks based implementations of PCA exist e.g (Oja 1989, Sanger 1989, Rubner & Tavan 1989, Foldiák 1989, Kung & Diamantaras 1990, Leen 1991). Karhunen and Joursensalo (1995) discuss various generalizations of neural network implementations of PCA.

In this section, we describe the PCA encoding and derive its least squares optimality. We describe implementations of PCA and discuss some of its shortcomings.

2.1.1 The PCA encoding

PCA projects a data vector x ($x \in \mathcal{R}^n$) onto the m ($m < n$) dimensional linear subspace spanned by the *leading eigenvectors*[1] of the data covariance matrix $\Sigma = E[(x-\mu)(x-\mu)^T]$, where $\mu = E[x]$ and $E[\cdot]$ denotes an expectation with respect to x. PCA encodes x as

$$\begin{aligned} z &= f(x) = V(x-\mu) \\ &= \left(e_1^T(x-\mu), \ldots, e_m^T(x-\mu)\right)^T \quad , \end{aligned} \tag{2.3}$$

where V is an $m \times n$ matrix whose rows, e_i, are the *leading* m orthonormal eigenvectors of Σ, and z is the m dimensional encoding. The components of z are called the principal

[1] The leading m eigenvectors $\{e_i | i = 1, \ldots, m\}$ of a positive semi-definite matrix are the m eigenvectors which correspond to the m largest eigenvalues. The indices are assigned such that the corresponding eigenvalues in increasing order are given by $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$.

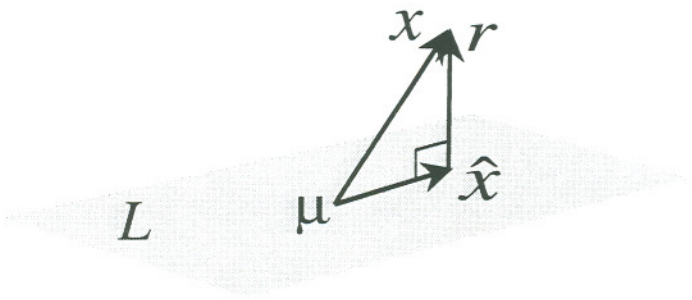

Figure 2.1: This figure shows the projection $\hat{x}$ of a vector x onto the subspace L. The PCA subspace L is spanned by the leading m eigen directions of Σ. Here $x = \hat{x} + r$, where r is the residual orthogonal to the subspace L.

components. PCA reconstructs x from z as

$$\hat{x} = g(z) = V^T z + \mu \ . \tag{2.4}$$

PCA can also be defined in terms of the eigenvectors of the data correlation matrix $R = E[xx^T]$ (e.g. see (Oja 1983)). Throughout this document, we will be using "PCA" to refer to the version which uses the covariance matrix.

From (2.3) and (2.4), the reconstructed vector $\hat{x}$ is given by,

$$\begin{aligned} \hat{x} &= g(f(x)) \\ &= \mu + V^T V\,(x - \mu) \\ &= \mu + \left(\sum_{i=1}^{m} e_i e_i^T\right)(x - \mu) \\ &= \mu + P\,(x - \mu) \ , \end{aligned} \tag{2.5}$$

where $P = \sum_{i=1}^{m} e_i e_i^T$ is a projection matrix[2] ($P \in \mathcal{R}^{n \times n}$) mapping any mean-subtracted vector $(x - \mu)$ to its projection vector on the subspace $L(e_1, \ldots, e_m)$ spanned by the leading m orthonormal eigenvectors of Σ. Figure 2.1 schematically shows the PCA projection onto the leading eigenspace. Figure 2.2 illustrates a 2-dimensional PCA of a 3 dimensional data set. The PCA defines a plane in $\mathcal{R}^3$. The low dimensional encoding

[2] P is symmetric ($P^T = P$), idempotent ($P^2 = P$) and $Py = 0$ for any $y \perp L(e_1, \ldots, e_m)$.

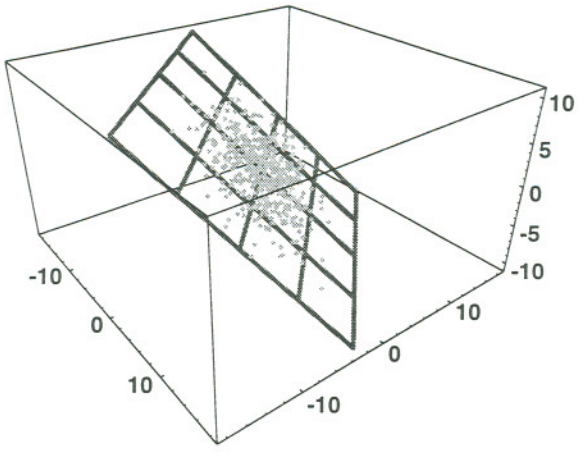

Figure 2.2: Global linear coordinates built by a 2 dimensional PCA for a 3-dimensional Gaussian distributed data. The grid lines denote the hyperplane spanned by the 2 leading eigenvectors of the data covariance matrix. The low dimensional encoding of a data point is given by coordinates *on the hyperplane* of the projection of the data point onto the hyperplane.

z is given by coordinates *on the subspace* $L\{e_1, \ldots, e_m\}$ of the projection of a given point x onto the plane. A comprehensive discussion of the PCA projection and its optimality properties is given in (Oja 1983). In the next subsection, we will outline a proof of the least squares optimality of PCA.

2.1.2 Least squares optimality of PCA

In this section, we will outline a proof of the least squares optimality of PCA. Any linear dimension reduction algorithm projects an input vector x onto an m dimensional linear manifold[3]. The mean squared error between x and $\hat{x}$ is given by the mean squared error induced by this projection. Suppose $\{a_1, \ldots, a_m, \ldots, a_n\}$ is a set of orthonormal basis vectors for $\mathcal{R}^n$ such that $\{a_1, \ldots, a_m\}$ is a set of orthonormal basis vectors for the linear manifold. The mean squared error $E[\|x - \hat{x}\|^2]$ is then given by the sum of the variances

[3]An m dimensional linear manifold $M(e_1, \ldots, e_m)$ centered on a is the set of n-vectors $\{z \mid z = a + b$ where b belongs to the subspace of $\mathcal{R}^n$ spanned by the vectors $e_1, \ldots, e_m\}$. Here, we assume that the linear manifold is centered on the mean.

of inputs along the directions $\{a_{m+1}, \ldots, a_n\}$:

$$\begin{aligned} E\left[\|x - \hat{x}\|^2\right] &= E\left[\left\|x - \mu - \sum_{i=1}^{m} a_i a_i^T (x - \mu)\right\|^2\right] \\ &= E\left[\left\|\sum_{i=m+1}^{n} a_i a_i^T (x - \mu)\right\|^2\right] \\ &= E\left[(x - \mu)^T \left(\sum_{i=m+1}^{n} a_i a_i^T\right)(x - \mu)\right] \\ &= \sum_{i=m+1}^{n} E\left[a_i^T (x - \mu)(x - \mu)^T a_i\right] \\ &= \sum_{i=m+1}^{n} a_i^T \Sigma a_i \ , \end{aligned} \tag{2.6}$$

where $E[\cdot]$ denotes an expectation with respect to the random vector x, and $\Sigma = E_x[(x - \mu)(x - \mu)^T]$ is the data covariance matrix.

Using $n-m$ Lagrange multipliers to enforce the orthonormality constraints $a_i^T a_i = 1$, we minimize the cost function

$$C = \sum_{i=m+1}^{n} a_i^T \Sigma a_i - \sum_{i=m+1}^{n} \lambda_i (a_i^T a_i - 1) \ , \tag{2.7}$$

with respect to a_i. Taking the derivative of C with respect to a_i and equating it to 0, we obtain

$$\Sigma a_i = \lambda_i a_i \ , \tag{2.8}$$

for $i = m+1, \ldots, n$. Therefore, $a_{m+1}, \ldots, a_n$ are orthonormal eigenvectors of Σ with corresponding eigenvalues $\lambda_{m+1}, \ldots, \lambda_n$. Substituting (2.8) in (2.6), we obtain

$$\begin{aligned} E\left[\|x - \hat{x}\|^2\right] &= \sum_{i=m+1}^{n} a_i^T \Sigma a_i \\ &= \sum_{i=m+1}^{n} a_i^T \lambda_i a_i \\ &= \sum_{i=m+1}^{n} \lambda_i \ . \end{aligned} \tag{2.9}$$

Thus the mean squared error is the sum of $n-m$ eigenvalues of Σ, and is minimized when $\lambda_{m+1}, \ldots, \lambda_n$ are the smallest eigenvalues of Σ and a_i are the corresponding orthonormal

eigenvectors. This implies that the optimal linear dimension reduction algorithm would project x onto the linear manifold spanned by the *leading* m eigenvectors of Σ. This is precisely what PCA does. Thus PCA is optimal in the least squares sense.

PCA also optimizes other criteria (Oja 1983). The principal components are mutually uncorrelated and are the components along the directions of maximum variance for the n-vector x. In chapter 4, we derive the principal components as the maximum likelihood signal estimate in a probabilistic signal-plus-Gaussian noise model. We will now describe our implementations of PCA.

2.1.3 Implementations of PCA

In this subsection, we describe implementations of PCA. Let $\mathcal{X} = \{x^1, \ldots, x^N\}$ denote a training set of N data points, where $x^i \in \mathcal{R}^n$, $i = 1, \ldots, N$. Suppose we are reducing data from n to m dimensions. We estimate μ and Σ by the sample mean

$$\hat{\mu} = \frac{1}{N} \sum_{I=1}^{N} x^I \quad \text{and}$$

the sample covariance matrix

$$\hat{\Sigma} = \frac{1}{N} \sum_{I=1}^{N} (x^I - \hat{\mu})(x^I - \hat{\mu})^T$$

respectively. We reduce $\hat{\Sigma}$ to a tridiagonal form using the Householder algorithm (Golub & van Loan 1983, Press, Flannery, Teukolsky & Vetterling 1987). We then use the QL algorithm with implicit shifts (Golub & van Loan 1983, Press *et al.* 1987) to compute the orthonormal eigenvectors e_i of $\hat{\Sigma}$. Finally, we use (2.3) to obtain low dimensional encodings.

When the input dimension n is very large (of the order of hundreds or thousands), it is not feasible to compute and store $\hat{\Sigma}$. This is often the case, for instance, when using PCA for image analysis (Cottrell *et al.* 1987). Also, when n is very large, we often have insufficient data to estimate $\hat{\Sigma}$. In the extreme case when the number of input vectors, N, is less than n, $\hat{\Sigma}$ is singular and we can not compute e_i by diagonalizing $\hat{\Sigma}$.

Singular value decomposition (SVD) (Golub & van Loan 1983, Press *et al.* 1987) is a robust method for estimating the eigenvectors of $\hat{\Sigma}$ in such cases. Suppose X is an $N \times n$ matrix whose rows are the N mean-subtracted input samples $\{(x^I - \hat{\mu}) | I = 1, \ldots, N\}$. The $n \times n$ matrix $X^T X$ is proportional to the sample covariance matrix $\hat{\Sigma}$

$$\hat{\Sigma} = \frac{1}{N} X^T X \quad . \tag{2.10}$$

The SVD of an $N \times n$ matrix X factors X as the product of an $N \times n$ column orthonormal matrix U^T, an $n \times n$ diagonal matrix W with positive or zero elements, and an $N \times n$ row orthonormal matrix V

$$X = U^T W V \quad . \tag{2.11}$$

Since U and V are orthonormal matrices,

$$U U^T = V V^T = I_{n \times n} \quad , \tag{2.12}$$

where $I_{n \times n}$ is an $n \times n$ identity matrix.

Using (2.10) and the decomposition in (2.11), we compute

$$\begin{aligned} \hat{\Sigma} &= \frac{1}{N} X^T X \\ &= \frac{1}{N} V^T W U U^T W V \quad \text{using (2.11),} \\ &= \frac{1}{N} V^T W^2 V \quad \text{using (2.12),} \\ &= V^T \frac{W^2}{N} V \quad . \end{aligned} \tag{2.13}$$

Thus, the SVD of X diagonalizes $\hat{\Sigma}$ and we have an alternate procedure for computing the eigen-system of $\hat{\Sigma}$. We first compute X. We then use SVD to decompose X into the matrices U, W, and V as in (2.11). The orthonormal eigenvectors of $\hat{\Sigma}$ are given by the rows of V and the corresponding eigenvalues are given by the corresponding diagonal elements of the matrix $\frac{W^2}{N}$ (using (2.13)). The decomposition in (2.11) can always be done, no matter how singular X is. This is particularly useful when the input data is scarce (N is small). We implemented both the methods described above using routines from *"The Numerical Recipes in C"* (Press *et al.* 1987).

2.1.4 The inadequacies of PCA

As discussed in section 1.1.2, PCA is the optimal *linear* algorithm for minimizing the expected squared error between x and its reconstruction $\hat{x}$. However, the linear assumption may be a severe restriction for some data sets.

PCA identifies and removes only linear correlations among the input variables. The principal components (components of z; (2.3)) are uncorrelated. This *does not* imply that they are statistically independent. Since PCA only considers second order statistics (i.e. correlations) of the data, it is unable to capture any higher order structure (non-linear dependencies) of the data. A non-linear technique can exploit the statistical dependencies among input variables to remove redundancies not removed by de-correlating the data. Low dimensional PCA encodings incur high error in the presence of such non-linear dependencies.

Looking at this problem from another perspective, consider the case when the target dimension m is not specified and we are asked to reduce the data to some dimension m such that the error is smaller than a given threshold. Since PCA identifies only linear correlations, for any given error threshold, PCA may retain more dimensions (larger m) than a non-linear technique, whenever the data has non-linear dependencies.

Geometrically, an m dimensional PCA models the data by an m dimensional hyperplane. If the input data lies on or near a curved manifold[4] of dimension equal to, or greater than m, then PCA will incur a high error. In this case, non-linear algorithms which model the input data by a projection onto a (curved) manifold will incur a lower error than PCA.

To illustrate with an example, consider data distributed on the surface of a hemisphere. Figure 2.3 shows the data and a two dimensional hyperplane defined by the PCA projection. Though the data set is three dimensional, it can be described using just two coordinates. The dependencies between the three Cartesian coordinates are non-linear

[4] A set M is a k dimensional manifold if it is locally *diffeomorphic* to $\mathcal{R}^k$. A smooth map $f : X \to Y$ is a diffeomorphism if it is one to one and onto, and if the inverse map $f^{-1} : Y \to X$ is also smooth. Thus, M is a k-dimensional manifold, if "resembles" $\mathcal{R}^k$ in the neighborhood of any point.

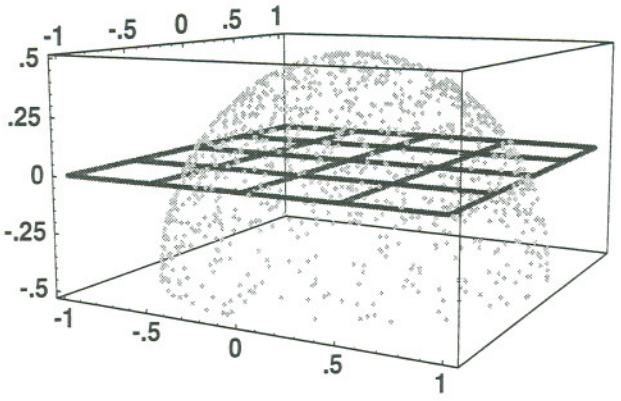

Figure 2.3: Global linear coordinates built by a 2 dimensional PCA for data distributed on the surface of a hemisphere. The PCA model is inaccurate because the input variables (Cartesian coordinates) have a non-linear functional dependence on each other.

and hence beyond the capability of a two dimensional PCA to detect. However, a non-linear algorithm should be able to capture this and obtain an accurate two dimensional encoding. In the next section, we describe auto-associative five layer networks which are capable of capturing such non-linear structure in the data.

2.2 Five layered auto-associative neural networks

Five layered auto-associative neural networks build a global *non-linear* model of the data. These nets reduce dimension by projecting the data onto a curved manifold of a smaller dimension than the input dimensionality.

In this section, we first describe auto-associative neural networks and the relation between three layered auto-associative networks and PCA. We then describe the motivation for using five layered networks for dimension reduction. We discuss our implementations of the network learning algorithms and finally we discuss some of the shortcomings of using this approach for dimension reduction.

2.2.1 Auto-associative neural networks

An *artificial neural network* is a data processing model which consists of a set of weighted connections between layers of data processing units known as nodes or units or neurons. In this document, we will consider only *feedforward* neural networks trained by supervised learning. Each network has distinguished *input* and *output* nodes. All other nodes are called *hidden nodes*. Each node of the network (except the input nodes) computes a weighted sum of its inputs and may further compute a non-linear function of the weighted sum. The resulting outputs of nodes are passed along the network connections to other nodes. Thus, the output nodes compute a well defined function of the inputs. The network connections (numerical weights) are trained to learn this function based on examples of input-output pairs. For further details and references to neural network based data modelling, the reader is referred to the book by Hertz, Krogh and Palmer (1991) and references therein.

Auto-associative neural networks are feedforward neural networks with a bottleneck layer trained to learn an identity mapping. Auto-associative neural networks can perform dimension reduction. These networks have at least one bottleneck layer, with a fewer number of nodes than the input dimensionality. These networks have the same number of output and input nodes, and are trained to reproduce the input variables at the output nodes. The networks are are trained to minimize the expected squared distance $E[\|x - \hat{x}\|^2]$ between the inputs x and the outputs $\hat{x}$. In trying to generate an identity mapping, the networks are forced to obtain a compact representation of the inputs in the bottleneck layer. Figure 2.4 shows a five layer auto-associative network. Here, the activations of the third layer nodes form a low dimensional encoding of the inputs.

Funahashi (1990) showed that the dimension reduction capability of *three* layered auto-associative networks (with or without non-linearities) does not exceed the capability of linear techniques (like PCA). Bourlard and Kamp (1988) showed that the encodings obtained by three layered auto-associative networks are equivalent to those obtained by PCA. Oja (1991) showed that five layer auto-associative networks can, in principle,

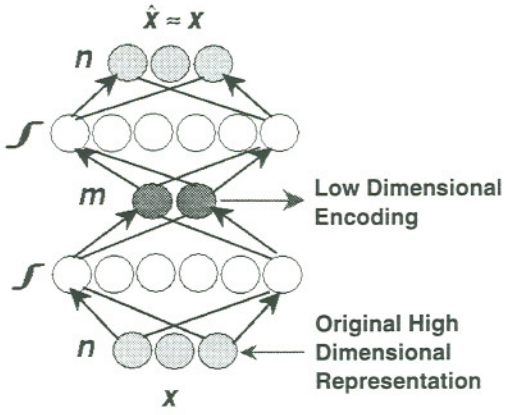

Figure 2.4: A five layer feedforward auto-associative network. The network tries to reconstruct the inputs at the output layer. The activations of the bottleneck (third) layer form the low dimensional encoding. This network can perform a non-linear dimension reduction from n to m dimensions.

perform a non-linear dimension reduction. In theory, these nets can compute any continuous mapping from the inputs to the bottleneck layer (Hornik, Stinchcombe & White 1989, Funahashi 1989), and another mapping from the bottleneck layer to the output layer.

2.2.2 Five layered networks (FLNs) for dimension reduction

Auto-associative five layered networks (FLNs) are capable of exploiting non-linear dependencies among the input variables to generate more accurate encodings than PCA. FLNs have been used for the analysis of simulated chemical batch reaction data (Kramer 1991), for the learning of psychological color attributes from surface spectral reflectance data (Usui, Nakauchi & Nakano 1991), and for image compression (Namphol, Arozullah & Chin 1991).

For reducing dimension from n to m ($m < n$), a five layered feedforward neural network has a bottleneck layer containing m nodes (see Figure 2.4). The network has n nodes in the input and output layers and is trained to perform an identity mapping on its inputs (auto-association). The second and the fourth layers are known as the *mapping*

layers since they determine the mapping $f(\cdot)$ from the input layer to the bottleneck layer and the mapping $g(\cdot)$ from the bottleneck layer to the output layer. For simplicity, we will only consider networks with an equal number of nodes in both the mapping layers. The activations of the mapping layer nodes are bounded by a sigmoidal non-linear function. For our purposes, a sigmoid is a monotone, continuous function with asymptotes of -1 and 1. The activations of the third layer form the m dimensional encoding z of the input data. The activations of the fifth layer form the n dimensional reconstructed input $\hat{x}$. The network is trained to minimize the mean squared error,

$$\mathcal{E} = E[\|x - \hat{x}\|^2] \ , \tag{2.14}$$

between the inputs and the corresponding reconstructions.

Oja (1991) gives a theoretical analysis of dimension reduction using auto-associative FLNs. Suppose there are q nodes in both the mapping layers of an FLN. Let $F : \mathcal{R}^n \rightarrow \mathcal{R}^q$ denote the FLN mapping from the input nodes to the first hidden layer. Oja showed that if we consider the activations of the second layer nodes as a modified q dimensional input vector $F(x)$, then the weights of the second through fourth layers can define cascaded PCA encoding and decoding functions for the modified input vector $F(x)$. In this case, the activations of the third layer nodes form the m dimensional PCA encoding of the q dimensional vector $F(x)$ and the activations of the fourth layer nodes form the PCA reconstruction of $F(x)$. The weights of fourth to fifth layer of the FLN define a mapping $FI : \mathcal{R}^q \rightarrow \mathcal{R}^n$, which can approximate an inverse mapping to F, if it exists. Thus, FLNs are, in principle, capable of generating non-linear encodings of the input vectors.

An FLN learns an m dimensional surface (the range of the mapping f from the inputs to the bottleneck layer nodes) which best represents the data, i.e minimizes the squared error described above. The reconstructed input $\hat{x}$ is a projection of x onto this surface and the low dimensional encoding z is the coordinates *on the surface* of the projection.

If we vary the activations of the m nodes of the bottleneck layer of a trained network, the propagated activations of the fifth layer represent the corresponding coordinates in the n dimensional space. This is illustrated in Figure 2.5 which shows a three to two

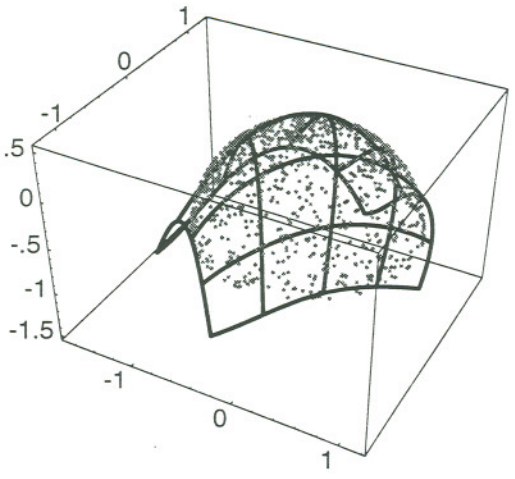

Figure 2.5: Global curvilinear coordinates built by a five layer network for data distributed on the surface of a hemisphere. When the activations of the representation layer are swept, the outputs trace out the curvilinear coordinates shown by the solid lines.

dimension reduction using a FLN. The data consists of points distributed on the surface of a hemisphere (this is the same data plotted in Figure 2.3). The grid lines in Figure 2.5 represent two dimensional coordinate axes on the curved surface learned by the FLN. A given data point x is projected onto the surface. The projection is given by $\hat{x}$ and the coordinates of the projection on the surface is given by z.

2.2.3 Implementation details for FLNs

We implemented the FLNs described above using an equal number of nodes and a sigmoid activation function

$$s(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

for both the mapping layers. We trained the FLNs using conjugate gradient descent, the BFGS algorithm and stochastic gradient descent. We computed the gradient of the cost function using the backpropogation algorithm (Rumelhart *et al.* 1986).

A learning algorithm is said to be in *batch mode* when each weight update uses information from all the data points in the training set. The algorithm is said to *on-line*, or *stochastic*, if each weight update is based on a single data point from the training set.

The batch mode algorithms compute the exact gradient at each step, but they can incur large training times when the number of input vectors is large and some of the vectors are redundant. In such situations, a stochastic algorithm will train significantly faster, though the gradient estimate at each step is inexact and noisy. This stochastic noise can be beneficial in helping the network jump out of local minima. We implemented a stochastic gradient descent (SGD) algorithm with a momentum term (Hertz *et al.* 1991) and an annealed learning rate (Darken & Moody 1991).

We also implemented two batch mode optimization schemes: conjugate gradient descent (CGD) and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm. Let p denote the total number of weights. CGD (Press *et al.* 1987) finds p mutually conjugate directions in the weight space. These directions are such that minimization along one of them does not spoil the previous minimization along the other directions. CGD requires only $O(p)$ storage and reaches the minimum in p steps if the cost function is quadratic in the weight space. BFGS (Press *et al.* 1987) makes use of the second order Hessian information of the cost function. The BFGS algorithm iteratively estimates the inverse Hessian and uses this curvature information to find new directions for line minimizations. BFGS requires $O(p^2)$ storage which make it infeasible for training large neural networks. We implemented CGD and BFGS using routines from *"The Numerical Recipes in C"*. We report results with all three algorithms in chapter 5.

2.2.4 Shortcomings of FLNs

Though, in principle, FLNs are capable of obtaining accurate low dimensional encodings, in practice, FLNs often produce encodings with a high error and take a long time to train. A global model may be inappropriate for some data sets. Consider a data set which has a different structure in different parts of the input space. For example, Figure 2.6(a) shows data distributed on an orbit asymptotic to the Lorenz attractor (Guckenheimer & Holmes 1983). The three dimensional data can be well described *locally* using only two dimensions, in most of the input space. However, the data is perhaps more accurately described as three dimensional in the region where the two wings of the attractors meet.

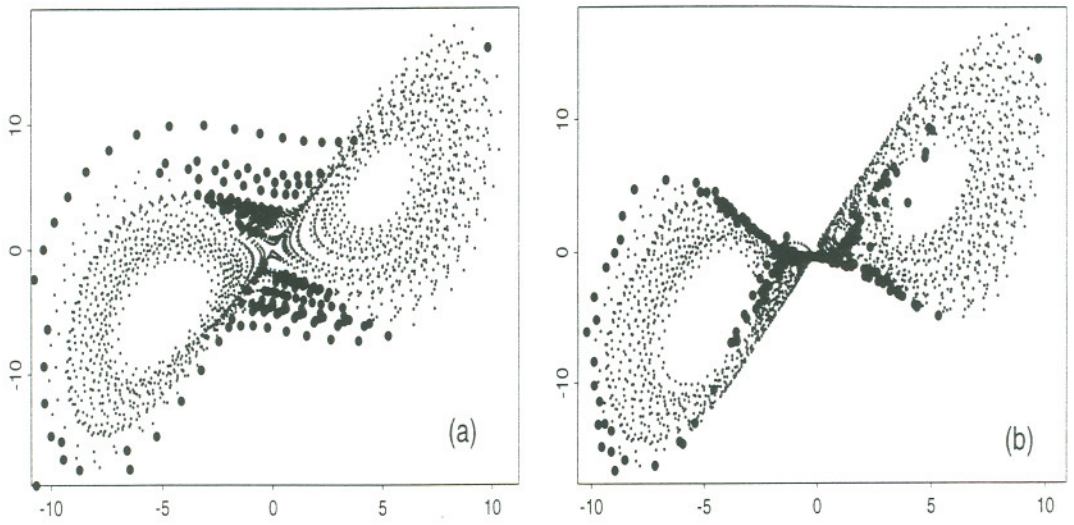

Figure 2.6: The Lorenz attractor data set : (a) data distributed on an orbit asymptotic to the attractor and (b) reconstructed data set from a two dimensional encoding obtained by an FLN with a configuration 3-45-2-45-3. The darker points represent the input points for which the FLN had its worst squared error.

Thus the data has a different structure of the data and different (local) dimensionality in different regions of the input space. A global model will have to compute a highly complicated two dimensional manifold to describe the data accurately. This process can be very inefficient.

Figure 2.6(b) shows the Lorenz data in Figure 2.6(a) reconstructed from a two dimensional FLN encoding. We notice that the model is quiet inaccurate near the regions where the wings meet. Moreover, the inaccurate fit extends well into the wings. Local models can generate more accurate fits for this type of data sets, since an inaccurate fit in one (local) region of the input space does not effect the fit in other regions of the input space for local models. Local models are not constrained to build a smooth m dimensional surface to fit the data.

Global models are also inefficient to train. Five layer networks have four layers of weights and hence a large number of parameters, even for moderate input dimensionality. Thus, FLNs can be memory intensive and can require a long time to train.

To summarize, global models can be inefficient and time consuming when the data lies near complex convoluted manifolds. Local techniques may provide more accurate and more compact encodings in this situation. In the next chapter, we will describe local linear algorithms for dimension reduction.

Chapter 3

Local linear models for dimension reduction

In this chapter, we present local linear algorithms for dimension reduction, which partition the input space and build separate low dimensional coordinate systems in local regions. Suppose we are reducing dimension of a data set from n to m ($m < n$). The goal is to find encoding functions $f : \mathcal{R}^n \rightarrow \mathcal{R}^m$ and decoding functions $g : \mathcal{R}^m \rightarrow \mathcal{R}^n$, such that the mean squared error in reconstructing an input vector x from $f(x)$,

$$\mathcal{E} = E\left[\| x - g(f(x)) \|^2 \right]$$

is minimized. Principal components analysis (PCA; section 2.1) and five layered auto-associative networks (FLNs; section 2.2) build global models for the encoding and decoding functions. Global models can be inefficient (harder to train, larger error) when the data has different structure in different regions of the input space (e.g. see the discussion in section 2.2.4). Building simple linear models in local regions of the input space can be faster and more accurate.

We propose the following procedure.

1. Partition $\mathcal{R}^n$ into Q disjoint regions $\{R_1, \ldots, R_Q\}$.

2. Do a separate linear projection onto an m dimensional space in each "local" region.

Thus, our encoding function is

$$f(x) = \begin{cases} f_1(x) & \text{if } x \in R_1. \\ \vdots & \\ f_Q(x) & \text{if } x \in R_Q. \end{cases} \quad , \tag{3.1}$$

where the local functions $f_1(\cdot), \ldots, f_Q(\cdot)$ are linear. This basic procedure defines a class of local linear algorithms depending upon the specific method used to implement the two steps. We use a vector quantizer (VQ) to partition $\mathcal{R}^n$ in the first step, and PCA to obtain m dimensional coordinates in the second step. We refer to this model as **VQPCA** (Kambhatla & Leen 1993, Kambhatla & Leen 1994). Bregler and Omohundro use a similar procedure based on local PCA for "learning non-linear constraint surfaces from the data" (1994) and for "manifold learning" for nonlinear image interpolation (1995). Hinton *et al* (1995) incorporate image invariance information to a local PCA model (similar to above) for handwritten digit recognition.

In the next section, we describe vector quantizers (VQs); algorithms for partitioning the input space into disjoint regions (also called cells). We define a VQ, discuss optimality criteria for VQs and algorithms for training VQs. Unconstrained VQs with a large number of cells incur prohibitive computation and memory requirements. We describe efficient algorithms for partitioning the input space using a hierarchical multi-stage or a tree structured VQ, which trade-off a loss in performance with a reduction in complexity.

In section 3.2, we describe several algorithms for implementing the VQPCA model, which differ in the method used to obtain the partition. We first describe an approach using a VQ with Euclidean distance. We then describe a new distortion measure called "reconstruction distance" which is optimal for performing a local PCA. We present a VQPCA training algorithm which uses a VQ with reconstruction distance. Finally, we present algorithms which use multi-stage and tree structured VQs to implement VQPCA.

3.1 Vector quantization (VQ)

Vector quantization (Gersho 1979, Gersho & Gray 1992) is a classical technique for signal coding and data compression. A vector quantizer (VQ) approximates a vector x by one of a predetermined set of prototype vectors called *reference vectors* or *codebook vectors*. With each VQ, there is an associated *distance measure* or *distortion measure*.

An n-dimensional vector quantizer of size Q and distortion measure $d(x, \mu_i)$ defines a mapping $g : \mathcal{R}^n \rightarrow \mathcal{C}$, from a vector in $\mathcal{R}^n$, onto a finite set $\mathcal{C} = (\mu_1, \ldots, \mu_Q)$, containing Q reference vectors, where $\mu_i \in \mathcal{R}^n$, $i = 1, \ldots, n$. The set $\mathcal{C}$ is called the *codebook*. For *Voronoi* or *nearest neighbor* quantizers[1], $g(\cdot)$ maps a vector x to the reference vector in $\mathcal{C}$ which is the closest to x with respect to $d(x, \cdot)$. Thus,

$$g(x) = \mu_{w(x)} \ , \tag{3.2}$$

where

$$w(x) = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} \ d(x, \mu_k) \ . \tag{3.3}$$

The mapping $g(\cdot)$ partitions $\mathcal{R}^n$ into Q disjoint regions or *cells*. The ith cell is defined as

$$\begin{aligned} R_i &= \{x \in \mathcal{R}^n : g(x) = \mu_i\} \\ &= \{x \in \mathcal{R}^n : d(x, \mu_i) \leq d(x, \mu_j)\} \ . \end{aligned} \tag{3.4}$$

Thus, a VQ builds a piece-wise constant model of the data. It partitions the input space into a set of regions and approximates each region by a reference vector.

VQ performs lossy data compression since the index $w(x)$ is sufficient to obtain $g(x)$, the VQ approximation to x (3.2). The *rate* of a VQ, $R = (\log_2 Q)$, measures the number of bits needed to represent x. The rate R is typically much smaller than the number of bits needed to specify the components of x. However, the compression is lossy since $d(x, g(x))$ is, in general, greater than 0. Plotting the expected distortion D induced by a

[1] In this document, we will restrict our discussion to nearest neighbor quantizers.

VQ as a function of the rate R (i.e an $R(D)$ curve) is a way of studying the compression performance of a VQ (Gersho & Gray 1992, Cover & Thomas 1991).

The performance of VQ coding is measured by the expected distortion between x, and $g(x)$.

$$\begin{aligned} D &= E[d(x, g(x))] \\ &= E[\min_{k=1}^{Q} d(x, \mu_k)] \ . \end{aligned} \tag{3.5}$$

where $E[\cdot]$ denotes an expectation with respect to x. The goal of VQ training algorithms is to find the codebook $\mathcal{C}$ which minimizes the distortion D (3.5).

3.1.1 Optimality criteria for a VQ

In this section, we describe the necessary conditions for a VQ to achieve optimal performance (minimal D). Given a codebook $\mathcal{C}$, the optimal partition employs a *nearest neighbor* encoding rule, generating the cells

$$R_i = \{x \in \mathcal{R}^n : d(x, \mu_i) \leq d(x, \mu_j) \text{ for all } j \neq i\} \ .$$

From (3.4), we see that every nearest neighbor VQ generates an optimal partition for a given codebook.

The optimal codebook $\mathcal{C}$ for a given partition is given by the *centroid condition*. Gersho and Gray (1992) define the generalized centroid, cent(R), of any set $R \in \mathcal{R}^n$ as "that vector y (it it exists) which minimizes the distortion between a point $x \in R$ and y, averaged over the probability distribution of x",

$$\begin{aligned} y^* &= \text{cent}(R) \\ &= \text{argmin}_y \ E[d(x, y) \,|\, x \in R] \ . \end{aligned} \tag{3.6}$$

For the squared error distortion measure, the centroid is simply the arithmetic mean of the data in the set R. The *centroid condition* states that, for a given partition $\{R_i; i = 1, \ldots, N\}$, the optimal reference vectors satisfy

$$\mu_i = \text{cent}(R_i) \ .$$

Another condition for optimality is that the collection of points equidistant (with respect to $d(\cdot,\cdot)$) from at least two reference vectors should have probability 0. All three optimality conditions listed above are necessary conditions. It is widely believed that they are sufficient for *local optimality*[2] (Gersho & Gray 1992), although no general theoretical derivation of this exists. In the next section, we will describe an iterative algorithm for training a VQ, which is based on the optimality conditions listed above.

3.1.2 The generalized Lloyd algorithm (GLA) for training a VQ

VQ training algorithms aim to minimize the expected distortion D (3.5) with respect to the codebook vectors μ_i based on a training set $\mathcal{X} = \{x^1, \ldots, x^N\}$ of N input vectors. The conditions for optimality discussed above are the basis for an iterative VQ training algorithm. We first initialize the VQ with random codebook vectors and compute the corresponding optimal partition. We then compute the optimal codebook for the new partition, and re-compute the optimal partition for the new codebook. We iterate these two steps till convergence.

This iterative codebook improvement algorithm for training a VQ is called the generalized Lloyd algorithm (GLA) (Gersho & Gray 1992). It is also known as the *k-means algorithm* after MacQueen (MacQueen 1967) who studied it as a statistical clustering technique, and as the LBG algorithm in the signal processing literature after Linde, Buzo and Gray (1980) who give a detailed description of the algorithm for data compression applications is presented. Using the terminology in Gersho and Gray (1992), we shall refer to it as the GLA since it is a direct generalization of Lloyd's treatment in 1957 (Lloyd 1982).

The GLA for training a VQ with Q cells and distortion measure $d(x,\cdot)$ is given below.

1. Start with an initial codebook $\mathcal{C}_1$. Initialize the iteration counter, $k = 1$.

[2] A VQ is locally optimal if every small perturbation of the codebook does not lead to a decrease in D. It is globally optimal if there exists no other codebook that gives a lower values of D (Gersho & Gray 1992).

2. **Partition::** Compute the optimal partition of the training vectors, given a codebook. Given a codebook $\mathcal{C}_k = \{\mu_1, \ldots, \mu_Q\}$, partition $\mathcal{X}$ into Q disjoint sets as

$$R_i = \{x \in \mathcal{X} : d(x, \mu_i) \leq d(x, \mu_j);\ \text{all } j \neq i\} \ . \tag{3.7}$$

   This partition assigns a training vector to the cell which corresponds to the nearest (in the $d(x, \mu_i)$ sense) codebook vector. If more than one codebook vectors have the minimum distance to an input vector, use a consistent tie-breaking strategy. For example, among the tied candidate cells, assign x to the set R_j for which j is smallest. Let N_j denote the number of vectors in the set R_j.

3. **Centroids computation::** Compute the optimal codebook for a given partition of the training set vectors. Given a partition of $\mathcal{X}$ into Q sets $R_i, i = 1, \ldots, Q$, compute the generalized centroid for each cell R_i as

$$\mu_i = \operatorname*{argmin}_{\mu} \frac{1}{N_i} \sum_{x \in R_i} d(x, \mu) \ . \tag{3.8}$$

   Let the new codebook be $\mathcal{C}_{k+1} = \{\mu_1, \ldots, \mu_Q\}$ where μ_i are the newly computed centroids.

4. Compute the distortion D_{k+1} (3.5) using the codebook $\mathcal{C}_{k+1}$. If the fractional change in error $(D_k - D_{k+1})/D_k$ is below some specified threshold, then stop. Otherwise, set $k + 1 \rightarrow k$ and go to step 2.

Each iteration of GLA reduces the average distortion D_k or leaves it unchanged (Lemma 11.3.1 in (Gersho & Gray 1992)), and D_k converges in a finite number of iterations (Lemma 11.3.2 in (Gersho & Gray 1992)). We can use any distortion measure $d(\cdot, \cdot)$, as long as we can analytically compute the corresponding generalized centroids.

The GLA is a *batch mode* algorithm, since each parameter update requires access to the entire training set. This can be very time consuming when the number of input vectors is large and some of the vectors are redundant. In such situations, *stochastic* algorithms can be faster. Stochastic algorithms perform parameter updates based on one randomly selected vector from the training set. In the next section, we describe a

class of neural network based learning algorithms for training VQs called *competitive learning*, which are stochastic versions of GLA.

3.1.3 Competitive learning algorithms

In this section, we will present stochastic versions of the GLA called *competitive learning* algorithms (Ahalt, Krishnamurthy, Cheen & Melton 1990, Hertz *et al.* 1991). Stochastic algorithms can be faster than batch-mode algorithms (like GLA) and can potentially jump out of local optima of the cost function due to the inherent stochastic noise. By default, competitive learning algorithms are for the Euclidean distance measure. A comparison of different competitive learning algorithms is given in (Ahalt *et al.* 1990).

A *competitive learning network* (Ahalt *et al.* 1990, Hertz *et al.* 1991) with Q neural units corresponds to a VQ with Q cells and the weight vectors of the network are the reference vectors μ_i. In each iteration, a randomly picked input vector x is presented to all the neural units, each of which computes the distortion between its reference vector and x. The unit with the smallest distortion is designated a "winner", and its reference vector $\mu_w(n)$ is updated as

$$\mu_w(n+1) = \mu_w(n) + \epsilon(x - \mu_w(n)) \ ,$$

where n denotes the iteration number and ϵ is a learning rate which is annealed as learning progresses. All other reference vectors are left unchanged. We iteratively present randomly chosen input vectors to the network until the network converges (the reference vectors stop changing above some threshold).

In the above algorithm, sometimes neural units can get under-utilized (Ahalt *et al.* 1990). This can lead to situations where some of the reference vectors never "win". A *frequency sensitive competitive learning* (FSCL; (Ahalt *et al.* 1990)) keeps track of how frequently each unit is the winner, and uses this information to modify the distortion measure $d(\cdot,\cdot)$ as follows. Let $u_i(n)$ denote the total number of times unit i has been the winner till the nth iteration. The modified distortion measure is given by

$$\hat{d}(x, \mu_i(n)) = d(x, \mu_i(n)) * u_i(n) \ .$$

The update rule and the training algorithm is exactly identical to standard competitive learning; the only difference being the distortion measure used.

A stochastic algorithm related to competitive learning algorithms is Kohonen's self-organizing feature map (KSFM) (Kohonen 1988). Kohonen initially formulated KSFM to illustrate the formation of topological feature maps in the brain, though it is often used to train a VQ. The KSFM updates the winning reference vector μ_w for a given input x *and* all other reference vectors μ_j which are in a topologically defined neighborhood $N(w)$ associated with unit w. The neighborhood is shrinked as learning progresses. Thus, the update rule is

$$\mu_j(n+1) = \mu_j(n) + \epsilon(x - \mu_j(n)) \text{ for all } j \in N(w),$$

for all reference vectors in the neighborhood $N(w)$, and all other reference vectors are left intact. Both the learning rate ϵ and the neighborhood $N(w)$ are annealed as learning progresses. The update rule given above is iterated for randomly presented input patterns until convergence (the reference vectors stop moving above some threshold).

All of the above algorithms are stochastic and avoid accessing all the training data vectors for each parameter update. This is useful when the training data is huge and parameter updates are relatively inexpensive. However, batch algorithms like GLA may be preferable when each parameter update incurs a lot of computation (e.g. this is the case when the distortion measure contains covariance matrices).

We have presented batch (GLA) and stochastic (competitive learning) algorithms for training a VQ. For all of these algorithms, the training time is dominated by distance computations of the VQ. When the number of cells is large, training a VQ can incur prohibitive training times and memory requirements. Moreover, we may not have enough data points to obtain good estimates of the reference vectors. It is possible to train VQs with large codebook sizes by applying constraints to the structure of the VQ codebook. These methods compromise performance, but significantly reduce the computational complexity of VQ. In the next two sections, we describe structurally constrained VQ algorithms.

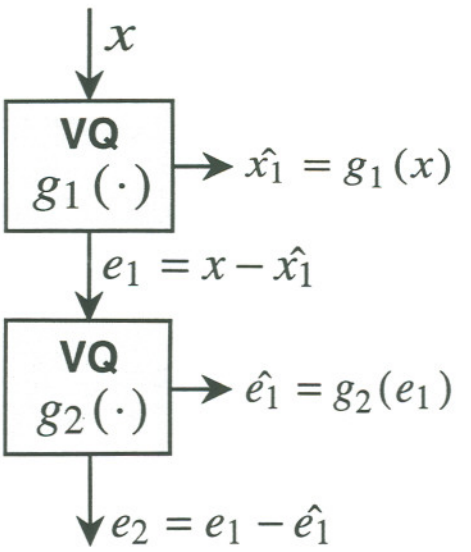

Figure 3.1: Figure showing the structure of a two-stage MSVQ. The first stage VQ quantizes an input vector x as $g_1(x)$. The second stage VQ quantizes the residual e_1 of the first stage ($e_1 = x - \hat{x}_1$) as $g_2(e_1)$. The MSVQ approximates x as $\hat{x}_1 + \hat{e}_1$. An MSVQ with l stages continues the process of quantizing residuals from previous stages for l stages.

3.1.4 Multi-stage VQ

An important technique for reducing the computational complexity of a VQ is *cascaded* or *residual* or *multi-stage* VQ (MSVQ) (Juang & Jr. 1982, Barnes & Frost 1990, Frost, Barnes & Xu 1991, Gersho & Gray 1992). The idea is to build the partition of the input space in cascaded stages, each of which uses a VQ. The first stage performs a crude quantization into a relatively small number of cells. A second stage quantizer uses the error vectors (called *residuals*) between the input vectors and their first stage VQ approximations (closest codebook vectors) as its inputs. This process is repeated; the second stage residuals are used as inputs for a third stage VQ and so on. Thus, we obtain a series of approximations to the original input vectors which are better with increasing number of stages. Figure 3.1 illustrates this process for a two stage MSVQ.

Training several small quantizers is computationally more efficient than training a single large unconstrained VQ. For example, training a two stage MSVQ with Q cells

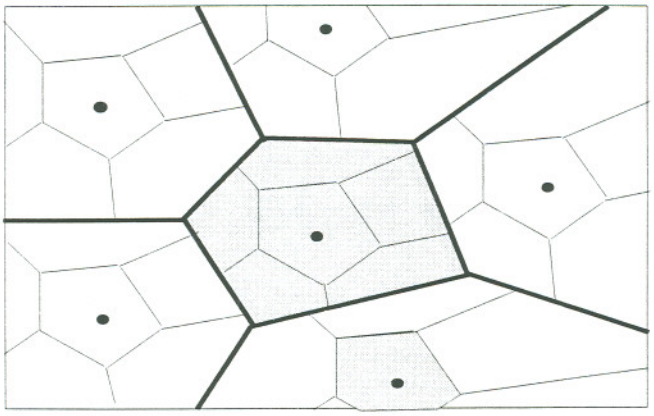

Figure 3.2: Figure showing a partition into Q^2 cells induced by a two-stage MSVQ with Q (here $Q = 6$) cells at each stage. The bold lines indicate the partition into Q cells induced by the first stage VQ. The shaded region in the center is one cell of the first stage partition. The narrow lines indicate the sub-partition of each first stage cell into Q sub-cells induced by the second stage VQ. The shaded region at the bottom left of the figure is one of the Q^2 cells in the second stage partition. Note the **identical sub-partitions** within all first stage VQ cells.

at each stage effectively produces Q^2 cells, though finding the closest reference vector at each stage involves comparisons with only Q vectors. If we had trained an unconstrained VQ with Q^2 cells, finding the closest reference vector would incur a comparison with Q^2 vectors. This is computationally much more expensive, especially when Q is large.

However, the Q^2 effective cells of the MSVQ are *constrained* since the second stage quantizer pools together the residuals from all the first stage cells. Figure 3.2 schematically shows a partition of $\mathcal{R}^2$ into Q^2 cells induced by a two-stage MSVQ with Q cells at each stage. The bold lines in the figure represent the partition induced by the first stage VQ. The residuals from *all* first stage cells are pooled together and quantized by the second stage VQ. This implies that the sub-division of each first stage cell into Q sub-cells is *identical* (in Figure 3.2, note the identical sub-partitions within all first stage cells). The constraint imposed by the identical sub-partitions within first stage cells can increase the average distortion induced by a two-stage MSVQ with Q cells at each stage, with respect to an unconstrained VQ with Q^2 cells.

MSVQ provides a tradeoff between a decrease in computational complexity and some loss in accuracy. For applications with a paucity of data, MSVQ allows us to train quantizers with a larger number of cells than an unconstrained VQ, since MSVQ has a much smaller number of parameters.

Let $\mathcal{X} = \{x^1, \ldots, x^N\}$ denote a training set. For an l stage MSVQ with Q cells at each stage, the MSVQ training algorithm is as follows:

1. Set the stage counter $s = 1$. Let $\mathcal{X}_s = \mathcal{X}$.

2. s^{th} **stage clustering::** Train a VQ with Q codebook vectors and a distance measure $d(x, \mu_i)$ for the data set $\mathcal{X}_s$ using the GLA (see section 3.1.2). Let the resulting codebook be $\mathcal{C}_s = \{\mu_1^s, \ldots, \mu_n^s\}$.

3. s^{th} **stage residuals::** Compute the residuals of $\mathcal{X}_s$ with respect to $\mathcal{C}_s$. The residuals are the error vectors between the input vectors $x^I \in \mathcal{X}_s$ and the closest (with respect to $d(\cdot, \cdot)$) codebook vectors from $\mathcal{C}_s$,

$$e_s^I = x^I - \mu_w^s \ .$$

where w is given by

$$w = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} \ d(x^I, \mu_k^s) \ .$$

Set the training data set $\mathcal{X}_{s+1}$ for the $(s+1)^{th}$ stage to be the residual vectors of the s^{th} stage. Increment the stage counter $s = s + 1$.

4. Repeat steps 2 and 3 until we have trained VQs for l stages, i.e. until $s > l$.

5. For each data point x, compute the indices w_i, $i = 1, \ldots, l$ of the closest codebook vectors for each of the l stages as follows. For the first stage VQ, compute the index of the cell

$$w_1 = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} \ d(x, \mu_k^1) \ ,$$

and the residual vector

$$e_1 = x - \mu_{w_1}^1 \ .$$

For stage i, $i = 2, \ldots, l$, compute the index of the cell as

$$w_i = \underset{k=1}{\overset{Q}{\operatorname{argmin}}}\ d(e_{i-1}, \mu_k^i) \ ,$$

and the residual vector as

$$e_i = e_{i-1} - \mu_{w_i}^i \ .$$

After computing the closest reference vectors at each stage, an MSVQ maps an input vector x to one of Q^l effective reference vectors as follows:

$$g(x) = \sum_{j=1}^{l} \mu_{w_j}^j$$

For transmission or storage purposes, the indices of the closest codebook vectors at each stage $(w_1, \ldots, w_l)$. are sufficient to recover $g(x)$.

Note that, we can use any distortion measure for the VQs, provided we can compute the generalized centroid for the GLA. In the next section, we will describe another algorithm for reducing the VQ search complexity, which can generate codebooks with a lower distortion than MSVQ.

3.1.5 Tree structured VQ

A tree structured vector quantizer (TSVQ; (Gersho & Gray 1992)) avoids the high computational cost of building an unconstrained VQ and tries to build more accurate encodings than a MSVQ. A TSVQ trains several VQs in tree structured stages. Suppose we are building a TSVQ with a depth d and a branching factor Q. We start at the root of the tree (level 1) and train a VQ with Q cells. We partition the training set into Q subsets using the codebook at the root. We train a *separate* VQ with Q cells for each of the training subsets and partition each of these subsets into Q sub-subsets. We continue this process for d levels of the tree. The final partition is determined by the reference vectors at the leaves (terminal nodes) of the tree. Thus, a TSVQ of depth d and a branching factor of Q effectively partitions the input space into Q^d cells. Figure 3.3 illustrates a two level TSVQ with a branching factor Q. A given input vector is compared with the Q reference vectors at each level of the tree. The nearest (with respect

to a distance measure $d(\cdot,\cdot)$) codebook vector determines which of the Q paths to take while descending the tree to the next level. The process continues until we reach a leaf of the tree.

A TSVQ of depth d and a branching factor of Q (which generates Q^d cells) is computationally more efficient than an unconstrained VQ with Q^d cells, since, in order to determine the cell membership of an input vector, the TSVQ needs only $d * Q$ comparisons, while the unconstrained VQ needs Q^d comparisons. However, TSVQ increases the storage requirements, since we need to store the codebook vectors at each node of the tree. Also, the effective partition built by a TSVQ is sub-optimal since the final placement of the last level reference vectors is not optimized based on on *all* of the data.

For an effective partition into the same number of cells, a TSVQ incurs more computation than an MSVQ since we need to train more VQs. However, the resulting partition may be more accurate (lower distortion) than that generated by an MSVQ. Compare a two stage MSVQ with Q cells at each stage and a two level TSVQ with a branching factor Q. Both these schemes generate Q^2 cells. The MSVQ partition is shown in Figure 3.2. As mentioned earlier, for the MSVQ, the sub-partitions within each cell of the first stage VQ are constrained to be identical. This is because the residuals from all cells of the first stage VQ are *pooled together* for the second stage VQ. In contrast, no such constraint exists for TSVQ, since we train Q separate VQs at the second level of the tree. Hence, the TSVQ partition *can* be more accurate than the MSVQ partition.

Suppose we are training a TSVQ of depth d, a branching factor Q, and a distortion measure[3] $d(\cdot,\cdot)$. Let $\mathcal{X} = \{x^1, \ldots, x^N\}$ denote a training set. The TSVQ training algorithm proceeds as follows:

1. Train a VQ with Q codebook vectors and a distance measure $d(\cdot,\cdot)$ for the data set $\mathcal{X}$ using GLA (see section 3.1.2). This induces a partition of the training data $\mathcal{X}$ into Q disjoint sets $\mathcal{X}_1, \ldots, \mathcal{X}_Q$.

2. Separately train a VQ of size Q with a distance measure $d(\cdot,\cdot)$ for each of the data

[3] As before, we can use any distortion measure as long as the generalized centroids are computable.

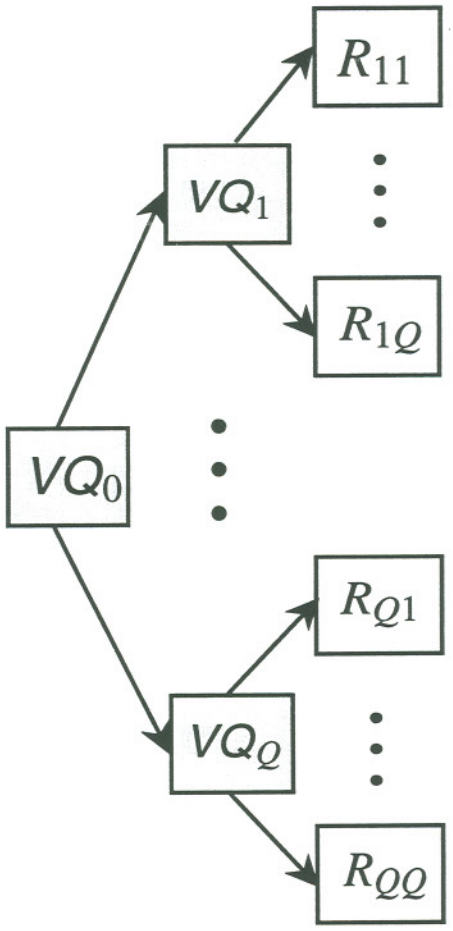

Figure 3.3: Figure showing the structure of a two level TSVQ with a branching factor Q. We first train a top level VQ (VQ_0) which partitions the training set into Q sets. We then train Q VQs ($VQ_1, \ldots, VQ_Q$) to sub-partition each of the Q training sub-sets into Q cells. The Q^2 cells $\{R_{11}, \ldots, R_{1Q}, \ldots, R_{Q1}, \ldots, R_{QQ}\}$ define the final partition of the TSVQ. The terminal nodes of the tree (the partitioned Q^2 cells) are shown in white.

sets $\mathcal{X}_i$ using GLA. We obtain codebooks for level 2 of the tree.

3. Partition each training set $\mathcal{X}_i$ into Q training subsets $\mathcal{X}_{ij}$ using the codebooks generated in step 2 and the distance measure $d(\cdot,\cdot)$. Use these new training sets $\mathcal{X}_{ij}$ to train Q^2 separate VQs for level 3.

4. Continue this process until we reach level d. Since each path from the root to a terminal node defines a cell of the VQ, the above process partitions the input data $\mathcal{X}$ into Q^d local regions. Let us denote these regions (cells) as $R_1, \ldots, R_{Q^d}$.

5. For each data point x, traverse the tree from top to bottom to determine the local region R_k. This is achieved by determining at each node (starting at the root), the closest codebook vector (say, the w^{th} vector) to x, and descending the tree along the w^{th} path from that node, until we reach a terminal node. A TSVQ approximates a vector x by

$$f(x) = \mu_w \ ,$$

where $x \in R_w$ in the final (dth level) partition at the leaf node.

In this section, we have described vector quantization, a piece-wise constant modelling technique which partitions the input space into a set of regions (cells) and approximates each local region with a reference vector. We have presented batch and on-line algorithms for training a VQ. We described multi-stage and tree structured quantizers which partition the input space in separate stages, thus reducing the computational complexity of a flat VQ.

In the next section, we will present our local linear algorithms for dimension reduction. These algorithms use a VQ to partition the input space into a set of regions. They then build a linear model using a PCA in each of the local regions.

3.2 Local linear dimension reduction

In this section, we present local linear dimension reduction algorithms which partition the input space using a VQ and build a PCA model in each local region. Suppose we

are reducing dimension of a data set from n to m dimensions. Global models (such as PCA and FLNs) can incur a large error when the data has different structure in different regions of the input space (e.g see the discussion in section 2.2.4). Building simple linear models in local regions of the input space can be faster and more accurate.

We propose the following model.

- Partition $\mathcal{R}^n$ into Q cells $\{R_1, \ldots, R_Q\}$ using a VQ with distortion measure $d(\cdot, \cdot)$.
- For $i = 1, \ldots, Q$, collect all training vectors in R_i and build an m dimensional PCA model.
- To reduce dimension of any vector $x \in \mathcal{R}^n$, compute its closest cell R_w (with respect to $d(\cdot, \cdot)$).
- Project x onto the PCA hyperplane for R_w.

We refer to this model as **VQPCA** (Kambhatla & Leen 1993, Kambhatla & Leen 1994). A VQPCA pieces together local linear coordinate patches (see Figure 3.4). The VQPCA encoding is $< w, z >$ where w is the index of the closest local region and z is the local m dimensional PCA encoding.

The VQPCA model is an extension of a standard VQ. VQPCA partitions the input space into a set of regions and approximates each region by a linear hyperplane (defined by PCA), while a standard VQ approximates each region by a codebook vector. Thus, VQPCA defines a *local linear* model of the data.

In this section, we describe algorithms for implementing the VQPCA model, which differ in the method used to obtain the partition. We first describe an approach using a VQ with Euclidean distance. We then describe a new distortion measure called "reconstruction distance" which is optimal for performing a local PCA. We present a VQPCA training algorithm which uses a VQ with reconstruction distance. Finally, we present algorithms which use multi-stage and tree structured VQs to implement VQPCA.

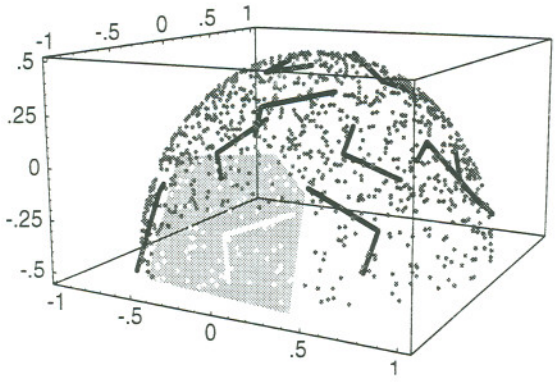

Figure 3.4: Local coordinates built by our algorithm (dubbed VQPCA) for data distributed on the surface of a hemisphere. The solid lines represent the two principal eigen-directions in each Voronoi cell. The region covered by one Voronoi cell is shown shaded.

3.2.1 Clustering with Euclidean distance

We first implement the VQPCA model using a VQ with Euclidean distance clustering. We use a GLA to train the VQ and partition the input space into Q regions. We then do a local PCA in each region. Suppose $\mathcal{X} = \{x^1, \ldots, x^N\}$ denotes a training set of N input vectors, where $x^I \in \mathcal{R}^n$, $I = 1, \ldots, N$. For a dimension reduction from n to m dimensions ($m < n$), and using Q local regions, the algorithm, dubbed VQPCA-Eucl, proceeds as follows:

1. Train a VQ with Q reference vectors $(\mu_1, \mu_2, \ldots, \mu_Q)$ and with the Euclidean distance measure using a GLA or competitive learning (see sections 3.1.2 and 3.1.3). This partitions the training set $\mathcal{X}$ into Q disjoint sets, $R_1, \ldots, R_Q$ (3.7). Let N_i denote the number of inputs mapped to R_i.

2. For each cell of the VQ, compute the local covariance matrix Σ_k defined as

$$\Sigma_k = \frac{1}{N_k} \sum_{x \in R_k} (x - \mu_k)(x - \mu_k)^T \quad .$$

Compute the orthonormal eigenvectors[4] $(e_{k1}, \ldots, e_{kn})$ of each Σ_k. Figure 3.4 shows two leading eigen-directions of disjoint VQ cells for data distributed on the surface of a hemisphere.

3. To reduce dimension of any vector $x \in \mathcal{R}^n$, compute the index w, of the cell closest (Euclidean distance) to x

$$w = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} \ \|x - \mu_k\|^2 \ , \tag{3.9}$$

and project x onto the leading m eigenvectors[5] of Σ_w to obtain the local linear coordinates

$$z = f_w(x) = (\ e_{w1} \cdot (x - \mu_w), \ldots, e_{wm} \cdot (x - \mu_w) \ ) \quad . \tag{3.10}$$

Thus, the VQPCA encoding function is a local linear function,

$$f(x) = \begin{cases} f_1(x) & \text{if } x \in R_1. \\ \vdots & \\ f_Q(x) & \text{if } x \in R_Q. \end{cases} \ , \tag{3.11}$$

where the local encoding functions are given by (3.10). After training, low dimensional encodings of any n dimensional data points can be obtained by using step 3 above. The encoding of x consists of the index w of the winning cell for x (3.9), together with the $m < n$ component vector z (3.10).

The decoding is given by

$$\hat{x} = g(x, w) \ = \ \mu_w \ + \ \sum_{i=1}^{m} \ z_i e_{wi} \ , \tag{3.12}$$

where μ_w is the reference vector for the cell w, and e_{wi} are the leading eigenvectors of the covariance matrix of the cell w. The distortion induced by a VQPCA encoding is

[4] We assume that the indices of the eigenvectors e_{ki} are ordered such that the corresponding eigenvalues are in decreasing order, $\lambda_{k1} \geq \ldots \geq \lambda_{kn}$ for all k.

[5] The leading m eigenvectors of a real symmetric matrix C are the m orthonormal eigenvectors which correspond to the m largest eigenvalues of C.

measured by the mean squared reconstruction error

$$\begin{aligned} D &= E\left[\|x-\hat{x}\|^2\right] \\ &= E\left[\left\|x-\mu_w-\sum_{i=1}^{m}(x-\mu_w)^T e_{wi}e_{wi}\right\|^2\right] \text{ using (3.12) and (3.10),} \\ &= E\left[\left\|\sum_{i=m+1}^{n}(x-\mu_w)^T e_{wi}e_{wi}\right\|^2\right] \text{ since } e_{wi},\ i=1,\ldots,n \text{ span } \mathcal{R}^n, \\ &= E\left[(x-\mu_w)^T\left(\sum_{i=m+1}^{n} e_{wi}e_{wi}^T\right)(x-\mu_w)\right] \\ &\equiv E\left[(x-\mu_w)^T P_w^{\perp}(x-\mu_w)\right], \end{aligned} \tag{3.13}$$

where $E[\cdot]$ denotes an expectation with respect to x, w is the index of the closest (Euclidean distance) cell of the VQ, given by (3.9), and $P_w^{\perp} \equiv \sum_{i=m+1}^{n} e_{wi}e_{wi}$ is a *local projection* matrix which projects the data onto a subspace orthogonal to the local m dimensional PCA hyperplane. Thus, the expected reconstruction error of VQPCA is given by the expected squared projection onto the subspace orthogonal to the local PCA hyperplane.

3.2.2 Clustering with "reconstruction distance"

The VQPCA-Eucl algorithm is not optimal because the clustering is done independently of the PCA projection. The goal is to minimize the expected error in reconstruction (3.13). We can realize this by using reconstruction error as the distortion measure for the VQ[6].

The expression for the VQPCA error in (3.13) suggests the distortion measure

$$d(x,\mu_k) = (x-\mu_k)^T P_k^{\perp}(x-\mu_k). \tag{3.14}$$

[6]We also implemented a VQ using the *Mahalanobis distance*

$$d(x,\mu_k) = (x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)$$

as its distortion measure. This is the clustering scheme suggested by a mixture of Gaussians probability model (see chapter 4) and previously used by Fraser and Dimitriadis (1993). In our experiments, we obtained a higher error than any of the schemes reported here.

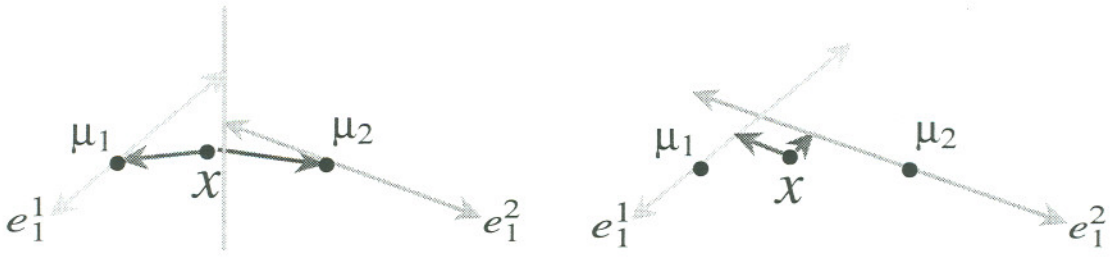

Figure 3.5: This figure illustrates the difference between clustering using Euclidean distance and clustering using the *reconstruction distance*. Suppose we want to determine the membership of a data point x to one of two regions. The figure on the left shows a partition based on Euclidean distance which is the distance to the centers μ_i. The point x gets assigned to R_1 using this distance measure. The figure on the right shows a partition based on reconstruction distance which is the distance to the local hyperplanes (in this case a line) defined by the local principal eigenvectors e_1^i; x now gets assigned to R_2. Cell assignment using the reconstruction distance is step 2 of the generalized Lloyd algorithm (GLA) described in section 3.2.2.

We call this the *reconstruction distance*. The reconstruction distance $d(x, \mu_k)$ is the error incurred in approximating x using *only* m local PCA coefficients (of cell k). It is the squared Euclidean distance to the linear manifold defined by the local PCA in the k^{th} local region. The distance is defined as the squared projection on the *trailing* $n - m$ eigenvectors of the covariance matrix. The Euclidean distance between x and μ_k is the sum of distances between x and μ_k along *all* eigen-directions of the covariance matrix, while the reconstruction distance is the sum of distances between x and μ_k along *only* the trailing $n - m$ eigen-directions.

Figure 3.5 illustrates the difference between Euclidean distance and the reconstruction distance. Suppose we want to determine the membership of a data point x to one of two cells[7] with reference vectors μ_1 and μ_2 and principal eigenvectors e_1^1 and e_1^2 respectively. Further suppose that we want to reduce the dimension of the data from two to one dimension. The data point x is closer to μ_1 than to μ_2 (Euclidean distance to μ_1 is smaller). However, x is closer to the local hyperplane (in this case, a line) defined by e_1^2

[7]Note that when considering membership of data points to VQ cells with respect to the reconstruction distance, the cells may not be connected sets.

than to the line defined by e_1^1. Thus, the point x gets assigned to cell R_1 using Euclidean distance and to cell R_2 using reconstruction distance. Thus the membership to cells defined by the reconstruction distance can be different than that defined by Euclidean distance. This is because the reconstruction distance does not penalize any spread of the data in the leading eigen-directions. This is exactly what is required since we retain all information in the leading directions while doing a PCA.

Suppose $\mathcal{X} = \{x^1, \ldots, x^N\}$ denotes a training data set. Using (3.13), where w is defined by the minimum reconstruction distance, the cost function to be minimized is

$$\mathcal{E} = \frac{1}{N} \sum_{I=1}^{N} \left[\min_{k=1}^{Q} (x^I - \mu_k)^T P_k^{\perp} (x^I - \mu_k) \right] \quad . \tag{3.15}$$

The VQPCA-Recon algorithm with Q cells for dimension reduction from n to m is:

1. Initialize μ_k to random input vectors from the training data set. Initialize Σ^k to identity matrices.

2. **Partition**:: Partition $\mathcal{X}$ into Q sets $R_1, \ldots, R_Q$, where

$$R_i = \{x \in \mathcal{X} : d(x, \mu_i) \leq d(x, \mu_j); \text{ all } j \neq i\} \ , \tag{3.16}$$

   using the reconstruction distance ($d(\cdot, \cdot)$ defined in (3.14)). Let N_i denote the number of elements in R_i. Figure 3.5 shows an example of a partition using reconstruction distance.

3. **Centroids computation**:: The generalized centroid (3.8) of a VQ cell R_k with respect to the reconstruction distance is

$$\mu_k = \underset{\mu}{\text{argmin}} \ \frac{1}{N_k} \sum_{x \in R_k} (x - \mu)^T \left(\sum_{i=m+1}^{n} e_{ki}(\mu) e_{ki}(\mu)^T \right) (x - \mu) \ , \tag{3.17}$$

   where we explicitly write the eigenvectors e_{ki} as functions of μ. In appendix A, we give a detailed derivation of the generalized centroids for reconstruction distance by computing the derivative of the right hand side of (3.17) with respect to μ (note

that the eigenvectors e_i are functions of μ). Equating the derivative of the right hand side of (3.17) with respect to μ (see appendix A) to 0, we obtain

$$-\frac{2}{N_k}\sum_{x\in R_k}\left(\sum_{i=m+1}^{n} e_{ki}(\mu)e_{ki}(\mu)^T\right)(x-\mu)=0 \ .$$

and hence,

$$\left(\sum_{i=m+1}^{n} e_{ki}(\mu)e_{ki}(\mu)^T\right)\mu=\left(\sum_{i=m+1}^{n} e_{ki}(\mu)e_{ki}(\mu)^T\right)\bar{x} \ , \tag{3.18}$$

where $\bar{x} \equiv \left(\frac{1}{N_k}\sum_{x\in R_k} x\right)$ is the arithmetic mean of elements of R_k. Thus, any vector μ whose projection along the trailing eigen-directions (with respect to μ) equals the projection of $\bar{x}$ along the trailing eigen-directions is a generalized centroid of R_k with respect to reconstruction distance. Here, we take $\mu = \bar{x}$, which is *a* solution of (3.18) to be the generalized centroid.

For each cell, compute μ_k as the mean of all the training set vectors assigned to that cell,

$$\mu_k = \frac{1}{N_k}\sum_{x\in R_k} x \ ,$$

and the covariance matrices Σ_k as,

$$\Sigma_k = \frac{1}{N_k}\sum_{x\in R_k}(x-\mu_k)(x-\mu_k)^T \ .$$

Next, compute the orthonormal eigenvectors e_i^k of each Σ_k.

4. Iterate Steps 2 and 3 until $\frac{\mathcal{E}_c-\mathcal{E}_{c+1}}{\mathcal{E}_c}$, the fractional change in the average reconstruction error (3.15) is below some specified threshold, where c is the iteration number.

5. To reduce dimension of any vector $x \in \mathcal{R}^n$, compute the index w, of the cell closest (*reconstruction distance*) to x

$$w = \underset{k=1}{\overset{Q}{\operatorname{argmin}}}\ (x-\mu_k)^T P_k^{\perp}\ (x-\mu_k) \ , \tag{3.19}$$

and project x onto the leading m eigenvectors of Σ_w to obtain the local linear coordinates

$$z = f_w(x) = (e_{w1} \cdot (x - \mu_w), \ldots, e_{wm} \cdot (x - \mu_w)) \quad .$$

The encoding of x consists of the index w and the m component vector z. The decoding is given by

$$\hat{x} \;=\; \mu_w \;+\; \sum_{i=1}^{m} z_i e_{wi} \quad . \tag{3.20}$$

The algorithm presented above is a generalized Lloyd algorithm (GLA (Gersho & Gray 1992); see section 3.1.2) with reconstruction distance as the distortion measure. Training in batch-mode avoids recomputing the eigenvectors e_{ki} (which depend on μ_k) after each input vector is presented. The algorithm directly minimizes the expected reconstruction error D (defined in (3.13)), where w is now defined by (3.19).

The training time of both the local linear algorithms described above (both VQPCA-Eucl and VQPCA-Recon) is dominated by the distance computations for the VQ. In the following two sections, we describe implementations of VQPCA which use multi-stage and tree structured VQs to partition the input space, to reduce the computational complexity.

3.2.3 Multi-stage clustering

Here, we use a multi-stage vector quantizer (MSVQ; see section 3.1.4) to partition the input space for VQPCA. Let $\mathcal{X} = \{x^1, \ldots, x^N\}$ be the training set, where $x^i \in \mathcal{R}^n, i = 1, \ldots, N$. For reducing dimension from n to m with an l stage MSVQ with Q cells for each stage, the **VQPCA-MS** algorithm proceeds as follows:

1. Train an l stage MSVQ with Q cells at each stage and with a distortion measure $d(\cdot, \cdot)$ using the data set $\mathcal{X}$ (see section 3.1.4 for details of training an MSVQ). Let $R_1, \ldots, R_Q$ denote the partition of the data set for the lth stage, $\mathcal{X}_l$, using the codebook of the l^{th} stage VQ. Let N_i denote the number of elements in R_i.

2. For each cell R_k, compute the local covariance matrices

$$\Sigma_k = \frac{1}{N_k} \sum_{r \in R_k} (r - \mu_k^l)(r - \mu_k^l)^T \ .$$

For each Σ_k, compute all the orthonormal eigenvectors e_{ki}, $i = 1, \ldots, n$, and the corresponding eigenvalues.

3. For any vector $x \in \mathcal{R}^n$, compute the indices w_i, $i = 1, \ldots, l$ of the closest codebook vectors for each of the l stages as follows. For the first stage VQ, compute the index of the cell

$$w_1 = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} \ d(x, \mu_k^1) \ ,$$

and the residual vector

$$r_1 = x - \mu_{w_1}^1 \ .$$

For stage i, $i = 2, \ldots, l$, compute the index of the cell as

$$w_i = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} \ d(r_{i-1}, \mu_k^i) \ ,$$

and the residual vector as

$$r_i = r_{i-1} - \mu_{w_i}^i \ .$$

The local coordinates are obtained by projecting r_l, the l^{th} stage residual onto the leading m eigen-directions of Σ_{w_l}

$$z = (e_{w_l 1} \cdot r_l, \ldots, e_{w_l m} \cdot r_l) \ .$$

The low dimensional encoding of VQPCA-MS consists of the indices of the closest codebook vectors at each stage $(w_1, \ldots, w_l)$ and the m dimensional vector z. The decoding is given by

$$\hat{x} = \sum_{j=1}^{l} \mu_{w_j}^j + \sum_{i=1}^{m} z_i e_{w_l i} \ ,$$

We can use either Euclidean distance or reconstruction distance for clustering in the algorithm described above. If we use Euclidean distance as the distance measure $d(\cdot, \cdot)$ for the VQ at all stages, we call the algorithm **VQPCA-E-MS**. An alternative

approach is to use Euclidean distance as the distance measure for the VQ for all but the last stage and train the l^{th} stage VQ using reconstruction distance as the distance measure. This can lower the overall error since clustering using reconstruction distance directly minimizes the local PCA error. We call this algorithm **VQPCA-R-MS**. In the next section, we describe algorithms for performing a tree structured codebook search for the VQ which can provide more accurate partitions than a multi-stage VQ.

3.2.4 Tree structured clustering

We now describe algorithms for training VQPCA using tree structured VQs (TSVQ; see section 3.1.5). Let $\mathcal{X} = \{x^1, \ldots, x^N\}$ denote a training set. For an n to m dimension reduction using a TSVQ with depth d and a branching factor Q, the **VQPCA-Tree** algorithm proceeds as follows:

1. Train a TSVQ of depth d, with a branching factor Q, and with a distortion measure $d(\cdot, \cdot)$ using the data set $\mathcal{X}$ (see section 3.1.5 for details of training a TSVQ). Since each path from the root to a terminal node defines a cell of the VQ, the above process partitions the input data $\mathcal{X}$ into Q^d local regions. Let us denote these regions (cells) as $R_1, \ldots, R_{Q^d}$. Let N_i denote the number of elements in R_i.

2. For each local region R_k, $k = 1, \ldots, Q^d$, compute the local data covariance matrix,

$$\Sigma_k = \frac{1}{N_k} \sum_{x \in R_k} (x - \mu_k)(x - \mu_k)^T \ .$$

   For each Σ_k, compute the orthonormal eigenvectors e_{ki}, $i = 1, \ldots, n$ and the corresponding eigenvalues.

3. For any vector $x \in \mathcal{R}^n$, traverse the tree from top to bottom to determine the local region R_k. This is achieved by determining at each node (starting at the root), the closest codebook vector (say, the w^{th} vector) to x, and descending the tree along the w^{th} path from that node, until we reach a terminal node. Once the local region R_w is determined ($x \in R_w$ for the partition in the leaf node), the local coordinates

Table 3.1: Notation used for the names of different algorithms presented in this chapter.

ALGORITHM NAME	CLUSTERING SCHEME	DISTANCE MEASURE
VQPCA-Eucl	Flat VQ	Euclidean
VQPCA-Recon	Flat VQ	reconstruction
VQPCA-E-MS	Multi-stage VQ	Euclidean
VQPCA-R-MS	Multi-stage VQ	reconstruction
VQPCA-E-Tree	Tree structured VQ	Euclidean
VQPCA-R-Tree	Tree structured VQ	reconstruction

are given by the projection onto the leading m eigen-directions of R_w

$$z = (e_{w1} \cdot (x - \mu_w), \ldots, e_{wm} \cdot (x - \mu_w)) \ .$$

The low dimensional encoding of VQPCA-Tree is given by the *indices of all the nodes in the path from the root* to the given terminal node (local region) and the m dimensional vector z. The decoding is given by

$$\hat{x} = \mu_w + \sum_{i=1}^{m} z_i e_{wi} \ .$$

We can use any distance measure in the algorithm described above. Using Euclidean distance at all nodes has the advantage that we do not have to store sets of eigenvectors for each node of the tree. We call this algorithm **VQPCA-E-Tree**. An alternative is to use the reconstruction distance for the VQ at all the nodes of the tree. This is very expensive storage wise, but can produce more accurate encodings. We call this algorithm **VQPCA-R-Tree**.

In this section, we have presented several algorithms for local linear dimension reduction. All of these algorithms build a piece-wise linear model of the data. Table 3.1 shows the nomenclature used to refer to these algorithms through the rest of the thesis. Unless specified otherwise, we will use the word **VQPCA** to refer to VQPCA-Recon, the algorithm which performs clustering with reconstruction distance. This notation will be used through the rest of the thesis. In the next section, we will summarize all the

algorithms discussed in this chapter. In chapter 4, we will present a probabilistic model for PCA and VQPCA based on a signal-plus-Gaussian noise model. In chapter 5, we will present experimental results with speech and image data, comparing all the algorithms discussed in this section with PCA and FLNs.

3.3 Summary

In this chapter, we described VQ, a piece-wise constant modelling technique which partitions the input space into a set of regions and approximates each region with a reference vector. We discussed optimality criteria for VQs and batch and on-line algorithms for training VQs. We also discussed multi-stage and tree structured VQs which can reduce the computational complexity of a VQ.

We then presented local linear algorithms for dimension reduction. All of these algorithms partition the input space into disjoint regions using a VQ. They then build local low dimensional coordinate systems using a PCA. Thus, they form a local linear model of the data.

The algorithms differ in the method of training the VQ. Clustering using Euclidean distance is the simplest, but can be suboptimal because the clustering is done independently of the PCA projection. Clustering using the *reconstruction distance*, which is the distance to the local PCA hyperplane, directly minimizes the VQPCA error. We presented a generalized Lloyd algorithm (GLA) to train the VQ with the new distortion measure. We also presented multi-stage and tree structured quantization schemes for reducing the computational cost of the above algorithms. These schemes trade-off some loss of accuracy for faster training and search times.

In the next chapter, we will present a probabilistic framework for PCA and VQPCA based on signal plus Gaussian noise models. We will show that PCA and VQPCA encodings approximate the maximum likelihood signal estimates for different signal-plus-noise models.

Chapter 4

A probabilistic framework for local linear dimension reduction

In this chapter, we present a probabilistic framework for principal components analysis (PCA; section 2.1) and our local linear dimension reduction algorithms (VQPCA; section 3.2.2) using signal plus Gaussian noise models. In this dissertation, we consider dimension reduction algorithms which reduce the dimension of a vector x ($x \in \mathcal{R}^n$) from n to m ($m < n$), such that the mean squared error between x and its reconstruction $\hat{x}$ is minimized (see chapter 1). The PCA reconstruction is a projection of x onto the leading m eigen-space of the covariance matrix of x. The VQPCA reconstruction is a *local* projection of x onto the the leading m eigen-space of the covariance matrix for the local region with the smallest "reconstruction distance" to x (the distance to the m dimensional hyperplane spanned the m leading eigen-directions).

In the next section, we show that the PCA projection of x approximates the maximum likelihood signal estimate for a signal-plus-noise model, where

- the signal density is a Gaussian with $n - m$ eigen-directions with negligibly small eigenvalues,
- the noise density is a spherically symmetric Gaussian, and
- the noise variance is much smaller than the signal variances.

We discuss the relation of this model to factor analysis.

In section 4.2, we show that, under certain conditions, a VQPCA projection of x *approximates* the maximum likelihood signal estimate for a probabilistic signal-plus-noise model, where

- the signal density is a mixture of Gaussians, each with $n-m$ eigen-directions with negligibly small eigenvalues,
- the noise density is a spherically symmetric Gaussian, and
- the noise variance is much smaller than the signal variances.

In section 4.2.4, we discuss the conditions under which this approximation holds. In section 4.2.4, we show the relation between VQPCA clustering using the reconstruction distance and maximum likelihood parameter estimation for the signal-plus-noise model.

4.1 A maximum likelihood model for PCA

In this section, we show that the PCA projection of x onto the leading eigen-space of the covariance matrix ((2.5) in section 2.1.1) is a maximum likelihood signal estimate of a Gaussian signal-plus-noise model. We will present the signal-plus-noise model for x and derive its correspondence to the PCA projection.

We assume that x is a sum of a signal vector $\tilde{x}$ and a noise vector ϵ,

$$x = \tilde{x} + \epsilon \tag{4.1}$$

where $x, \tilde{x}, \epsilon \in \mathcal{R}^n$, and $\tilde{x}$ lies very close to an m dimensional hyperplane. We assume that the signal and noise processes are statistically independent.

We assume that the signal density function $\tilde{p}(\tilde{x})$ is a multivariate n-dimensional Gaussian,

$$\tilde{p}(\tilde{x}) = \frac{1}{(2\pi)^{n/2}\sqrt{|\tilde{\Sigma}|}} \exp\left[-\frac{1}{2}(\tilde{x}-\mu)^T \tilde{\Sigma}^{-1}(\tilde{x}-\mu)\right] \tag{4.2}$$

where the covariance matrix $\tilde{\Sigma}$ has m statistical degrees of freedom with a large variance $\tilde{C}_i, i = 1, \ldots, m$, and $n-m$ degrees of freedom with a small variance $\delta_i, i = m+1, \ldots, n$.

Let $\tilde{\Sigma} = V^T \tilde{\Lambda} V$, where the rows of V are the orthonormal eigenvectors of $\tilde{\Sigma}$. The diagonal eigenvalue matrix $\tilde{\Lambda}$ (by model assumption) is given by

$$\tilde{\Lambda} = \begin{pmatrix} \tilde{C}_1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & \tilde{C}_m & 0 & \dots & 0 \\ 0 & \dots & 0 & \delta_{m+1} & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & \delta_n \end{pmatrix} . \tag{4.3}$$

We assume that $\delta_i, i = m+1, \dots, n$ are very small (later we will take the limit $\delta_i \to 0$). This implies that the signal is lying *very* close to an m dimensional hyperplane. Finally, we assume that the density function of ϵ is a spherically symmetric zero mean Gaussian with variance σ^2,

$$p_\epsilon(\epsilon) = \frac{1}{(2\pi)^{n/2}(\sigma^2)^{n/2}} \exp\left[-\frac{1}{2\sigma^2}\, \epsilon^T \epsilon\right] . \tag{4.4}$$

Based on the model described in (4.1-4.4), and using the independence of $\tilde{x}$ and ϵ, the input density function $p(x)$ is

$$\begin{aligned} p(x) &= \int d^n\tilde{x} \int d^n\epsilon \;\; p(x \,|\, \tilde{x}, \epsilon)\, \tilde{p}(\tilde{x})\, p_\epsilon(\epsilon) \\ &= \int d^n\tilde{x} \int d^n\epsilon \;\; \delta(x - \tilde{x} - \epsilon)\, \tilde{p}(\tilde{x})\, p_\epsilon(\epsilon) \\ &= \int d^n\tilde{x} \;\; \tilde{p}(\tilde{x})\, p_\epsilon(x - \tilde{x}) \\ &= \frac{1}{(2\pi)^{n/2}\sqrt{|\Sigma|}} \exp\left[-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right] , \end{aligned} \tag{4.5}$$

where $\Sigma = \tilde{\Sigma} + \sigma^2 I_{n\times n}$ has the orthonormal eigenvector matrix V and the diagonal

eigenvalue matrix Λ,

$$\Lambda = \begin{pmatrix} C_1 = \tilde{C}_1 + \sigma^2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & C_m = \tilde{C}_m + \sigma^2 & 0 & \dots & 0 \\ 0 & \dots & 0 & \delta_{m+1} + \sigma^2 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & \delta_n + \sigma^2 \end{pmatrix} . \tag{4.6}$$

To obtain the maximum likelihood estimate of the signal based on the observed pattern x and the model, we compute the $\tilde{x}$ which maximizes the *conditional signal likelihood* $p(\tilde{x} \mid x)$,

$$\begin{aligned} x^* &= \underset{\tilde{x}}{\text{argmax}}\ p(\tilde{x} \mid x) \\ &= \underset{\tilde{x}}{\text{argmax}}\ \frac{p_\epsilon(x - \tilde{x})\, \tilde{p}(\tilde{x})}{p(x)} \quad \text{using Bayes rule and the independence of } \tilde{x} \text{ and } \epsilon, \\ &= \underset{\tilde{x}}{\text{argmax}}\ p_\epsilon(x - \tilde{x})\, \tilde{p}(\tilde{x}) \ . \end{aligned} \tag{4.7}$$

We evaluate (4.7) by taking the derivative of the $p_\epsilon(x - \tilde{x})\tilde{p}(\tilde{x})$ with respect to $\tilde{x}$ and equating it to 0. We obtain

$$x^* = \mu + \sum_{i=1}^{m} \frac{\tilde{C}_i}{C_i} e_i e_i^T (x - \mu) + \sum_{i=m+1}^{n} \frac{\delta_i}{\delta_i + \sigma^2} e_i e_i^T (x - \mu) \ , \tag{4.8}$$

where $e_i, i = 1, \dots, n$ are the orthonormal eigenvectors of both Σ and $\tilde{\Sigma}$, and the indices are assumed to be ordered such that the corresponding eigenvalues are in a decreasing order as i increases. We take the limit $\delta_i \to 0$ for $i = m + 1, \dots, n$ and obtain

$$x^* = \mu + \sum_{i=1}^{m} \frac{\tilde{C}_i}{C_i} e_i e_i^T (x - \mu) \ . \tag{4.9}$$

Thus, the maximum likelihood signal estimate is a *scaled* projection of x onto the leading m eigen-space of the covariance matrix of x. The projection of x along e_i is scaled by $\frac{\tilde{C}_i}{C_i} = \frac{C_i - \sigma^2}{C_i}$. This scaling is a contraction since x has noise components (spherically distributed with variance σ^2) in *all* directions.

If we assume that the noise variance σ^2 is negligible compared to the signal variances, $\sigma^2 \ll \tilde{C}_i$ for $i = 1, \ldots, m$, (4.9) reduces to

$$x^* = \mu + \sum_{i=1}^{m} e_i c_i^T (x - \mu) , \tag{4.10}$$

the PCA projection ((2.5) in section 2.1.1). Thus, the PCA projection of x onto the leading m eigen-space of the covariance matrix of x is the maximum likelihood signal estimate (4.10) for our Gaussian signal-plus-noise model.

4.1.1 The relation between factor analysis and the maximum likelihood model for PCA

The above model is very similar in spirit to the "the basic factor-analytic model" used in *factor analysis* (Harman 1976, Gnanadesikan 1977, Dillon & Goldstein 1984). In this subsection, we show the relation between factor analysis and the probabilistic model described above. The basic model in factor analysis is

$$x = \Lambda f + \epsilon , \tag{4.11}$$

where $x, \epsilon \in \mathcal{R}^n$, $\Lambda \in \mathcal{R}^{n \times m}$, and $f \in \mathcal{R}^m$ for $m < n$. The components of f are called the *common factors*, the components of ϵ are called the *unique factors* and the components of the matrix Λ are called the *factor loadings*. The unique factors are assumed to be independent of the common factors and are assumed to have a zero mean and a diagonal covariance matrix

$$E[\epsilon\epsilon^T] = \Delta = \begin{pmatrix} \sigma_1^2 & 0 & \ldots & 0 \\ 0 & \sigma_2^2 & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & \sigma_n^2 \end{pmatrix} . \tag{4.12}$$

Factor analysis techniques (Harman 1976, Gnanadesikan 1977, Dillon & Goldstein 1984) estimate the factor loadings and values of unique factors from sample observations of the random vector x. This is similar to signal estimation for a signal-plus-noise model, where $\tilde{x} = \Lambda f$ denotes a "signal" vector and ϵ denotes a "noise" vector.

We now derive the maximum likelihood signal estimate for the factor analysis model. Let $\tilde{x} = \Lambda f \in \mathcal{R}^n$. Let $N(\mu, \Sigma)$ denote a multi-variate normal distribution with mean μ and covariance matrix Σ. We assume a signal plus noise model $x = \tilde{x} + \epsilon \equiv \Lambda f + \epsilon$, where $x, \tilde{x}, \epsilon \in \mathcal{R}^n$ and the noise vector ϵ and f are statistically independent. We assume that f is $N(\mu_f, I_{m\times m})$ (this is a standard assumption; see (Harman 1976, Gnanadesikan 1977)). Therefore, $\tilde{x}$ is $N(\mu = \Lambda\mu_f, \tilde{\Sigma} = \Lambda\Lambda^T)$. We assume that ϵ is $N(0, \Delta)$ (Harman 1976, Gnanadesikan 1977), where Δ is given by (4.12). The maximum likelihood signal estimate is

$$\begin{aligned} x^* &= \operatorname*{argmax}_{\tilde{x}} \; p(\tilde{x} \,|\, x) \\ &= \operatorname*{argmax}_{\tilde{x}} \; p_\epsilon(x - \tilde{x})\tilde{p}(\tilde{x}) \;\; . \end{aligned} \tag{4.13}$$

There is no solution to (4.13) unless $\tilde{\Sigma}$ is invertible. The factor analysis model assumes that $\tilde{\Sigma} = \Lambda\Lambda^T$, where $\Lambda \in \mathcal{R}^{n\times m}$ and $m < n$. Therefore, $\tilde{\Sigma}$ is rank m and hence not invertible. Thus, we *can not* obtain the maximum likelihood signal estimate starting from the factor analysis model. In order to do so, we need to assume that $\tilde{x}$ has variances in all eigen-directions as shown in section 4.1.

4.2 A maximum likelihood model for local linear dimension reduction

In this section, we derive the correspondence between the VQPCA algorithms for local linear dimension reduction (section 3.2.2) and a mixture of Gaussians signal-plus-noise model. We use a similar analysis to that given in section 4.1 to show how the VQPCA projection relates to the maximum likelihood signal estimate for the signal-plus-noise model. We also show the relation between clustering with reconstruction distance and maximum likelihood clustering under this model.

We first describe our mixture of Gaussians signal-plus-noise model. We derive the input density $p(x)$, a mixture of Gaussians. We derive the maximum likelihood signal

estimate for the signal-plus-noise model and discuss its relation to the VQPCA projection. Finally we discuss maximum likelihood parameter estimation for our model and its relation to VQ clustering using reconstruction distance (section 3.2.2).

4.2.1 The signal-plus-noise model

We model the vector x as a sum of a mixture of Gaussians distributed signal vector $\tilde{x}$, and a spherically symmetric Gaussian distributed noise vector ϵ. We assume that

$$x = \tilde{x} + \epsilon \ , \tag{4.14}$$

where $x, \tilde{x}, \epsilon \in \mathcal{R}^n$, and $\tilde{x}$ and ϵ are statistically independent. We assume that the signal density function $\tilde{p}(\tilde{x})$ is a *mixture* of Q multivariate n dimensional Gaussians,

$$\tilde{p}(\tilde{x}) = \sum_{k=1}^{Q} \frac{\alpha_k}{(2\pi)^{n/2}\sqrt{|\tilde{\Sigma}_k|}} \exp\left[-\frac{1}{2}(\tilde{x}-\mu_k)^T \tilde{\Sigma}_k^{-1}(\tilde{x}-\mu_k)\right] \ , \tag{4.15}$$

where all the component covariance matrices $\tilde{\Sigma}_k$ have m large variance directions with variance $\tilde{C}_i$, and $n-m$ small variance directions with variance δ_j. Thus, if we diagonalize $\tilde{\Sigma}_k$ as $\tilde{\Sigma}_k = V_k^T \tilde{\Lambda} V_k$, where the rows of V_k are the orthonormal eigenvectors of $\tilde{\Sigma}_k$, then the eigenvalue matrix $\tilde{\Lambda}$ is given by

$$\tilde{\Lambda} = \begin{pmatrix} \tilde{C}_1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & \tilde{C}_m & 0 & \dots & 0 \\ 0 & \dots & 0 & \delta_{m+1} & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & \delta_n \end{pmatrix} \ . \tag{4.16}$$

We assume that all the mixture components have *identical* covariance eigenvalues, though their eigenvectors may differ. This implies that $|\tilde{\Sigma}_k|$ is the same for all mixture components.

We assume that all δ_j are very small (later we will take the limit $\delta_i \to 0$). This implies that the signal is lying very close to an m dimensional manifold. We further assume that

the noise density $p_\epsilon(\epsilon)$ is given by a spherically symmetric zero mean Gaussian with variance σ^2

$$p_\epsilon(\epsilon) = \frac{1}{(2\pi)^{n/2}(\sigma^2)^{n/2}} \exp\left[-\frac{1}{2\sigma^2}\,\epsilon^T\epsilon\right] . \tag{4.17}$$

We assume that the noise variance σ^2 is much smaller than the signal variances, i.e $\sigma^2 \ll \tilde{C}_i$, for $i = 1, \ldots, m$.

4.2.2 The input density $p(x)$ given the signal-plus-noise model

Using (4.15), (4.17) and the independence of $\tilde{x}$ and ϵ, we derive the input density $p(x)$

$$\begin{aligned} p(x) &= \int d^n\tilde{x} \int d^n\epsilon \;\; p(x \,|\, \tilde{x}, \epsilon)\, \tilde{p}(\tilde{x})\, p_\epsilon(\epsilon) \\ &= \int d^n\tilde{x} \int d^n\epsilon \;\; \delta(x - \tilde{x} - \epsilon)\, \tilde{p}(\tilde{x})\, p_\epsilon(\epsilon) \;\; \text{using (4.14).} \\ &= \int d^n\tilde{x} \;\; \tilde{p}(\tilde{x})\, p_\epsilon(x - \tilde{x}) \\ &= \sum_{k=1}^{Q} \frac{\alpha_k}{(2\pi)^{n/2}\sqrt{|\Sigma_k|}} \exp\left[-\frac{1}{2}(x - \mu_k)^T\, {\Sigma_k}^{-1}\, (x - \mu_k)\right] , \end{aligned} \tag{4.18}$$

where the component covariance matrices $\Sigma_k = \tilde{\Sigma}_k + \sigma^2 I_{n\times n}$ have the same eigenvectors as $\tilde{\Sigma}_k$, and an eigenvalue matrix $\Lambda = \tilde{\Lambda} + \sigma^2 I_{n\times n}$,

$$\Lambda = \begin{pmatrix} C_1 = \tilde{C}_1 + \sigma^2 \approx \tilde{C}_1 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & C_m = \tilde{C}_m + \sigma^2 \approx \tilde{C}_m & & 0 & \ldots & 0 \\ 0 & \ldots & 0 & \delta_{m+1} + \sigma^2 \approx \sigma^2 & \ldots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \ldots & 0 & 0 & \delta_n + \sigma^2 \approx \sigma^2 & \end{pmatrix} . \tag{4.19}$$

Note that the determinant $|\Sigma_k| = \prod_{i=1}^{m} C_i \prod_{i=m+1}^{n} (\delta_i + \sigma^2) \approx \sigma^{2(n-m)} \prod_{i=1}^{m} C_i$ is identical for all mixture components.

A winner-take-all approximation of $p(x)$

To derive the relation to VQPCA, we assume the following.

- The mixing proportions (α_k) are equal, i.e. $\alpha_k = 1/Q \; \forall k = 1, \ldots, Q$,
- For any given x, the density $p(x)$ (4.18) is dominated by the largest term in the summation. Thus, we obtain a hard partition of the data from the Gaussian mixture model. Such a *probabilistic clustering* has been extensively used in the statistics community (e.g see (Lazarsfeld & Henry 1968, Ganesalingam & McLachlan 1979, Aitkin *et al.* 1981, Basford & McLachlan 1985, McLachlan & Basford 1988)). Nowlan (1991) uses the above assumption, called the *"winner-take-all"* (WTA) assumption, to show the relation between Gaussian mixture densities and VQ clustering. This relation was earlier suggested by Duda and Hart (1973). In appendix B, we summarize a derivation of the relation between Gaussian mixture models and VQ clustering. Basically, the WTA assumption assigns all the responsibility for an observation to the Gaussian component which best explains it.

Using these assumptions,

$$p(x) = \max_{k=1}^{Q} Z \exp\left[-\frac{1}{2}(x-\mu_k)^T \Sigma_k{}^{-1} (x-\mu_k)\right] \quad , \tag{4.20}$$

where $Z = 1/(Q(2\pi)^{n/2}\sqrt{\prod_{i=1}^{m} \tilde{C}_i}\sigma^{(n-m)})$. The *"winning"* (dominant) Gaussian for a given x is the Gaussian with the smallest Mahalanobis distance to x,

$$\begin{aligned} w &= \operatorname*{argmax}_{k=1}^{Q} Z \exp\left[-\frac{1}{2}(x-\mu_k)^T \Sigma_k{}^{-1} (x-\mu_k)\right] \\ &= \operatorname*{argmin}_{k=1}^{Q} (x-\mu_k)^T \Sigma_k{}^{-1} (x-\mu_k) \\ &\approx \operatorname*{argmin}_{k=1}^{Q} (x-\mu_k)^T \left(\sum_{i=1}^{m} \frac{e_{ki} e_{ki}^T}{C_i} + \sum_{i=m+1}^{n} \frac{e_{ki} e_{ki}^T}{\sigma^2}\right) (x-\mu_k) \quad , \end{aligned} \tag{4.21}$$

where e_{ki}, $i = 1, \ldots, n$ are the orthonormal eigenvectors of Σ_k.

4.2.3 The maximum likelihood signal estimate

We now derive the maximum likelihood signal estimate for a given x. We compute the $\tilde{x}$ which maximizes the conditional signal likelihood $p(\tilde{x} \mid x)$.

$$\begin{aligned} x^\star &= \underset{\tilde{x}}{\text{argmax}}\ p(\tilde{x} \mid x) \\ &= \underset{\tilde{x}}{\text{argmax}}\ \frac{p(x \mid \tilde{x})\, \tilde{p}(\tilde{x})}{p(x)} \text{ using Bayes rule.} \\ &= \underset{\tilde{x}}{\text{argmax}}\ \frac{p_\epsilon(x-\tilde{x})\, \tilde{p}(\tilde{x})}{p(x)} \text{ using (4.14).} \\ &= \underset{\tilde{x}}{\text{argmax}}\ p_\epsilon(x-\tilde{x})\, \tilde{p}(\tilde{x}) \ . \end{aligned} \tag{4.22}$$

We apply the winner-take-all (WTA) assumption to the mixture function $\{\max_{\tilde{x}}\ p_\epsilon(x-\tilde{x})\tilde{p}(\tilde{x})\}$. We assume that for a given x, the same mixture component (w^{th} term (4.21)) dominates the mixture for both $p(x)$ and $\{\max_{\tilde{x}}\ p_\epsilon(x-\tilde{x})\tilde{p}(\tilde{x})\}$. This is a reasonable assumption, since from (4.18),

$$p(x) = \int d^n\tilde{x}\ \ p_\epsilon(x-\tilde{x})\, \tilde{p}(\tilde{x}) \ .$$

The density $p_\epsilon(x-\tilde{x})$ is a spherically symmetric Gaussian with variance σ^2 which has a maximum value when $\tilde{x} = x$. If we assume that the noise variance is negligibly small, i.e $\sigma^2 \to 0$, then, $p(x) \approx \tilde{p}(x)$, and we can assume that the mixtures $p(x)$ and $\{\max_{\tilde{x}}\ p_\epsilon(x-\tilde{x})\tilde{p}(\tilde{x})\}$ are dominated by the same term (w^{th} term) for a given x.

Using these assumptions in (4.22), we get

$$\begin{aligned} x^\star &= \underset{\tilde{x}}{\text{argmax}}\ p_\epsilon(x-\tilde{x})\, \tilde{p}(\tilde{x}) \\ &\approx \underset{\tilde{x}}{\text{argmax}}\ p_\epsilon(x-\tilde{x})\, \frac{1}{Q(2\pi)^{n/2}\sqrt{\tilde{C}^m \delta^{(n-m)}}}\ \exp\left[-\frac{1}{2}(\tilde{x}-\mu_w)^T\, \tilde{\Sigma}_w^{-1}\, (\tilde{x}-\mu_w)\right] \\ &\text{where } w \text{ is the } \textit{winner} \text{ given by (4.21).} \end{aligned} \tag{4.23}$$

We evaluate (4.23) to obtain

$$x^\star = \mu_w\ +\ \sum_{i=1}^{m} \frac{\tilde{C}_i}{C_i} e_{wi} e_{wi}^T\, (x-\mu_w)\ +\ \sum_{i=m+1}^{n} \frac{\delta_i}{\delta_i+\sigma^2} e_{wi} e_{wi}^T\, (x-\mu_w) \ . \tag{4.24}$$

Taking the limit $\delta_i \to 0$, $i = m+1, \ldots, n$, we obtain,

$$x^\star \approx \mu_w\ +\ \sum_{i=1}^{m} \frac{\tilde{C}_i}{C_i} e_{wi} e_{wi}^T\, (x-\mu_w) \ . \tag{4.25}$$

Thus, the maximum likelihood signal estimate is a *scaled* projection onto the leading m eigen-space of the w^{th} Gaussian component of the mixture density $p(x)$. The projection along $e_{wi}, i = 1, \ldots, m$ is scaled by $\frac{\tilde{C}_i}{C_i} = \frac{C_i - \sigma^2}{C_i}$ since our model postulates noise components (spherically symmetric Gaussian noise with variance σ^2) in all directions.

If we assume that the noise variance σ^2 is much less than the signal variances, i.e $\sigma^2 \ll \tilde{C}_i$, $i = 1, \ldots, m$, we obtain

$$x^* \approx \mu_w + \sum_{i=1}^{m} e_{wi} e_{wi}^T (x - \mu_w) \quad . \tag{4.26}$$

Thus, the maximum likelihood signal estimate for our model under the above assumptions is a PCA projection onto the leading m eigen-space of the w^{th} Gaussian component of the mixture density $p(x)$. The index w is the index of the Gaussian component with the smallest Mahalanobis distance (4.21) to x.

4.2.4 The relation between VQPCA encoding and the maximum likelihood signal estimate

The maximum likelihood signal estimate is of the same form as the VQPCA projection $\hat{x}$ (3.20 in section 3.2.2). For VQPCA, the local region w is the Gaussian component with the smallest reconstruction distance (3.19) to x, which is the squared projection along only the trailing eigenvectors of the covariance matrix Σ_k,

$$w = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} (x - \mu_k)^T \left(\sum_{i=m+1}^{n} e_{ki} e_{ki}^T \right) (x - \mu_k) \quad . \tag{4.27}$$

For the maximum likelihood signal estimate (section 4.2.3), the local region w is defined as the Gaussian component with the smallest *Mahalanobis distance* to x,

$$w = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} (x - \mu_k)^T \left(\sum_{i=1}^{m} \frac{e_{ki} e_{ki}^T}{C_i} + \sum_{i=m+1}^{n} \frac{e_{ki} e_{ki}^T}{\sigma^2} \right) (x - \mu_k) \quad , \tag{4.28}$$

as derived in section 4.2.3. The Mahalanobis distance is the sum of squared projections along *all* the eigenvectors of Σ_w, where each projection is scaled by the corresponding eigenvalue. Thus, the VQPCA projection can be interpreted as a maximum likelihood

signal estimate for which excursions within the local m-dimensional leading eigen-space are cost-free.

The assignment of local regions using Mahalanobis distance and using reconstruction distance will be nearly the same whenever the Mahalanobis distance in (4.28) is dominated by the terms corresponding to the trailing eigen-directions. Thus, the VQPCA projection is also the maximum likelihood signal estimate whenever the inequality

$$\sum_{i=1}^{m} \frac{(x-\mu_w)^T e_{wi} e_{wi}^T (x-\mu_w)}{C_i} \ll \sum_{i=m+1}^{n} \frac{(x-\mu_w)^T e_{wi} e_{wi}^T (x-\mu_w)}{\sigma^2}$$

holds. This condition is equivalent to the inequality

$$\sum_{i=1}^{m} \frac{(x-\mu_w)^T e_{wi} e_{wi}^T (x-\mu_w)}{C_i} < k \sum_{i=m+1}^{n} \frac{(x-\mu_w)^T e_{wi} e_{wi}^T (x-\mu_w)}{\sigma^2} \tag{4.29}$$

where k is a very small constant ($k \to 0$).

Let $\lambda_1 = C_1, \ldots, \lambda_m = C_m, \lambda_{m+1} = \sigma^2, \ldots, \lambda_n = \sigma^2$ denote the eigenvalues (in decreasing order) of Σ_w. We change coordinates from x to y,

$$y = H(x-\mu_w) \equiv \begin{pmatrix} \frac{e_{w1}^T}{\sqrt{\lambda_1}} \\ \vdots \\ \frac{e_{wn}^T}{\sqrt{\lambda_n}} \end{pmatrix} (x-\mu_w) \ . \tag{4.30}$$

In the changed coordinates, (4.29) reduces to

$$\sum_{i=1}^{m} y_i^2 < k \sum_{i=m+1}^{n} y_i^2 \tag{4.31}$$

The probability mass of the data points violating (4.29) is the same as the probability mass of the data points violating (4.31). Therefore, the probability mass of the data points violating (4.29) is *independent of* C_i, $i = 1, \ldots, m$ and σ^2 and depends only upon the relative values of k, the input dimensionality n, and the reduced dimension m.

We performed Monte Carlo simulations to study the dependence of the fraction of points violating (4.31) on n and m. I am grateful to Steven Rehfuss for suggesting this approach, writing the initial code and help with the simulations. We performed Monte Carlo simulations sampling 500,000 data points uniformly distributed in a unit n

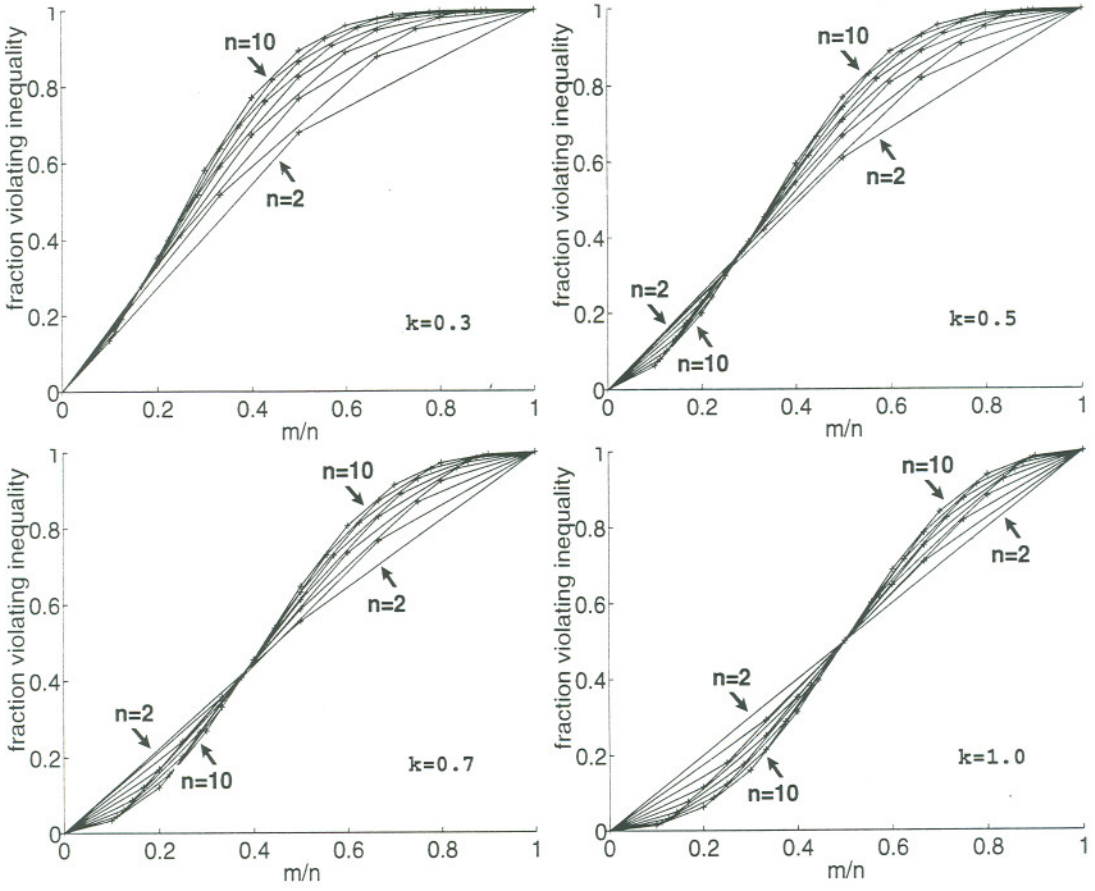

Figure 4.1: Plots showing the fraction of points violating the inequality (4.31) for 500,000 points uniformly sampled from a unit n dimensional hypersphere. Each plot is for a different value of k in (4.31) and shows 9 curves for different values of n. The horizontal axis plots the ratio m/n and the vertical axis plots the fraction of points violating (4.31).

dimensional sphere and recording the fraction of points for which (4.31) is violated. We performed experiments varying n from 2 to 10 and with all possible values of m (for a given n). Figure 4.1 plots m/n versus the fraction of points violating (4.31) for different values of k. Each plot in Figure 4.1 is for a different value of k. Each plot shows 9 curves, one for each n.

We notice from Figure 4.1 that, in general, the reduced dimension m should be much smaller than n for the condition (4.31) to hold most often. If we just consider the relative magnitudes of terms instead of dominance (i.e. considering the case with $k = 1.0$), the condition is violated least often when m is very small compared to n and when n is large. Thus, our Monte Carlo simulations suggest that the VQPCA projection approximates the maximum likelihood projection whenever m is much smaller than n and the approximation holds better for larger n (with fixed m/n).

4.2.5 Maximum likelihood clustering for the winner-take-all model

We will now derive maximum likelihood estimates of the parameters $\theta = \{\mu_k, e_k\}$ required to obtain the signal encoding in (4.26). The parameters C_i, $i = 1, \ldots, m$ and σ^2 are not considered here, since, by model assumption these are the same for all mixture components and they are not needed to compute the signal encoding (4.26). Let $\mathcal{X} = \{x^1, \ldots, x^N\}$ denote a training set of N data points. Assume that the samples in $\mathcal{X}$ are independent and identically distributed. The maximum likelihood parameter estimates can be obtained by maximizing the likelihood of the data set $\mathcal{X}$ as follows:

$$\begin{aligned}
\hat{\theta} &= \underset{\theta}{\operatorname{argmax}}\ p(\mathcal{X} \mid \theta) \\
&= \underset{\theta}{\operatorname{argmax}} \prod_{I=1}^{N} p(x^I \mid \theta) \ \text{ using i.i.d.} \\
&= \underset{\theta}{\operatorname{argmax}} \prod_{I=1}^{N} \max_{k=1}^{Q} Z \exp\left[-\frac{1}{2}(x^I - \mu_k)^T \Sigma_k{}^{-1} (x^I - \mu_k)\right] \ \text{ using (4.20),} \\
&= \underset{\theta}{\operatorname{argmin}} \sum_{I=1}^{N} \left[\min_{k=1}^{Q} (x^I - \mu_k)^T \left(\sum_{i}^{m} \frac{e_{ki} e_{ki}^T}{C_i} + \sum_{i=m+1}^{n} \frac{e_{ki} e_{ki}^T}{\sigma^2}\right) (x^I - \mu_k)\right] \quad (4.32)
\end{aligned}$$

after taking a log, and having discarded terms that do not depend on θ. The above expression for the data likelihood is the summed Mahalanobis distance to the closest (Mahalanobis distance) centroid or reference vector.

Thus, the maximum likelihood estimates of the mixture density parameters can be obtained by minimizing the summed Mahalanobis distance. If we assume, as discussed earlier, that the Mahalanobis distance is dominated by the terms corresponding to the trailing eigen-directions for most of the data ((4.29) and (4.31)) then the maximum likelihood parameter estimates are given by

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{I=1}^{N} \left[\min_{k=1}^{Q} (x^I - \mu_k)^T \left(\sum_{i=m+1}^{n} e_{ki} e_{ki}^T \right) (x^I - \mu_k) \right] \quad . \tag{4.33}$$

The cost function to be minimized above is the summed *"reconstruction distance"*. Thus we can approximate the maximum likelihood estimates by minimizing the summed reconstruction distance. This is precisely what the VQPCA algorithm does. The resulting parameter estimates can be considered to be maximum likelihood estimates provided (4.31) holds for most of the data points, or equivalently (as discussed in the previous section), m is much smaller than n.

4.3 Discussion

In this chapter, we have presented a probabilistic framework for PCA and VQPCA algorithms based on signal plus Gaussian noise models. We showed that the PCA projection of x is the maximum likelihood signal estimate for a signal-plus-noise model, where

- the signal density is a Gaussian with $n - m$ eigen-directions with negligibly small eigenvalues,
- the noise density is a spherically symmetric Gaussian and
- the noise variance is much smaller than the signal variances.

Though this model is related to a factor analysis model, we showed that we can not derive a maximum likelihood signal estimate directly from the factor analysis model.

We showed that a local PCA projection of x onto the leading eigen-space of the w^{th} local region approximates the maximum likelihood signal estimate for a signal-plus-noise model, where

- the signal density is a mixture of Gaussians, each with $n - m$ eigen-directions with negligibly small eigenvalues, and all component Gaussians have the same eigenvalues,
- the noise density is a spherically symmetric Gaussian,
- the noise variance is much smaller the the signal variances, and
- w is the index of the Gaussian component with the smallest Mahalanobis distance to x for the mixture density $p(x)$.

We showed that the VQPCA projection of x approximates the maximum likelihood signal estimate whenever the reconstruction distance approximates the Mahalanobis distance. We hypothesize, based on Monte Carlo simulations that the partitions generated by the two distance measures are nearly the same whenever $m \ll n$. Under this assumption, the parameter values obtained by clustering using reconstruction distance approximate the maximum likelihood parameter estimates.

In the next chapter, we compare PCA, FLNs and VQPCA for the tasks of speech and image dimension reduction, speech feature extraction and speech and image compression.

Chapter 5

An empirical comparison of dimension reduction algorithms

In the preceding chapters, we described several algorithms for dimension reduction. In this chapter, we present the results of experiments comparing these algorithms applied to real world speech and image data. We compare principal components analysis (PCA, section 2.1), five layered networks (FLNs, section 2.2), and our local linear algorithms (VQPCA, section 3.2) for speech and image dimension reduction and speech feature extraction. We also compare VQ and VQPCA for lossy compression of speech and image data.

First, we present results of experiments with the dimension reduction of spectral speech data and image pixel data. We compare the different algorithms based on the accuracy of reconstructions obtained from low dimensional encodings and the training times. The local linear algorithms obtain nearly half the error of FLNs or PCA while training much faster than FLNs.

We then present results showing the potential application of dimension reduction algorithms for speech feature extraction. We estimate formants (resonant frequencies of the vocal tract) of vowels from spectral features using the low dimensional encodings obtained by the dimension reduction algorithms. Finally, we compare VQ and VQPCA for lossy data compression. We compare rate/distortion $R(D)$ curves for speech and image compression. In general, VQ obtains the least distortion for a given bit rate.

5.1 Dimension reduction experiments

In this section, we apply PCA, five layer networks (FLNs), and VQPCA to dimension reduction of speech and images. We compare the algorithms using four performance criteria: training time, the distortion in the reconstructed signal, the number of floating point operations (FLOPS) required to encode an input vector (encode time) and the number of FLOPS required to reconstruct the input vector from the encoding (decode time).

Suppose we are reducing data from n to m dimensions where $m < n$. The distortion measure is the squared reconstruction error normalized by the data variance,

$$\mathcal{E} = \frac{\mathcal{E}_{recon}}{E[\,\|x-\mu\|^2\,]} = \frac{E[\,\|x-\hat{x}\|^2\,]}{E[\,\|x-\mu\|^2\,]}\,, \tag{5.1}$$

where $\mu = E[x]$ is the mean of the data, $E[\cdot]$ denotes an expectation with respect to x, and $\hat{x}$ denotes a reconstruction of x from an m dimensional encoding. The normalization gives us the fraction of the total data variance accounted by the m dimensional encodings. For example, $\mathcal{E} = 0.01$ indicates that 1% of the data variance is not accounted for by the m dimensional encodings. Let $\mathcal{X} = \{x^1, \ldots, x^N\}$ denote a test set of input vectors. We estimate (5.1) based on $\mathcal{X}$ as

$$\mathcal{E}_{norm} = \frac{\sum_{I=1}^{N} \|\, x^I - \hat{x^I}\,\|^2}{\sum_{I=1}^{N} \|x^I - \bar{x}\|^2}\,, \tag{5.2}$$

where $\bar{x} \equiv \frac{1}{N}\sum_{I=1}^{N} x^I$ is the arithmetic mean of the data set.

5.1.1 Experimental setup

For speech data, we trained PCA using the Householder reduction followed by QL algorithm (see section 2.1.3). For image data, our training set has fewer data points than the input dimensionality. Therefore, the auto-covariance matrix is singular and we cannot use Householder reduction. For image data, we trained PCA using singular value decomposition (SVD; see section 2.1.3).

We trained the FLNs using three optimization techniques: conjugate gradient descent (CGD), the BFGS algorithm (a quasi-Newton method (Press *et al.* 1987)), and stochastic

gradient descent (SGD). In order to limit the space of architectures, we only considered FLNs with the same number of nodes in both of the mapping (second and fourth) layers. See section 2.2 of chapter 2 for more details of the FLN training procedure.

We trained the local linear algorithms using Euclidean distance clustering (VQPCA-Eucl), reconstruction distance clustering (VQPCA-Recon), and multi-stage and tree structured clustering for both distance measures. We trained the VQ partition for VQPCA-Eucl using competitive learning (section 3.1.3; a stochastic learning algorithm) with an annealed learning rate (Darken & Moody 1991). We trained the VQ partition of VQPCA-Recon using a GLA (section 3.1.2; a batch-mode algorithm), since stochastic learning with reconstruction distance entails computing the covariance matrices and their eigenvectors after each randomly drawn input vector is presented. We implemented all the multi-stage and tree structured VQ algorithms using GLA. Further details of the clustering algorithms are given in section 3.2 of chapter 3. We will use the notation described in Table 3.1 to refer to different algorithms.

For determining the values of free parameters of different algorithms, we divided the data into training, validation and test sets. We varied the parameters over a range of values, and chose the value for which the validation set error was the least. We varied the number of nodes in the mapping layers (second and fourth layers) of the FLNs from 5 to 50 in increments of 5. Thus, we tried network configurations from $n-5-m-5-n$ to $n-50-m-50-n$. We picked the network configuration with the least validation set error for each optimization scheme (BFGS, CGD and SGD). We monitored the validation set error while training the FLNs and stopped training when the validation set error started to increase (*early stopping*; (Hertz *et al.* 1991)). For stochastic gradient descent (SGD), we used Darken and Moody's (1991) annealed learning rate schedule with a momentum (Hertz *et al.* 1991) term and stopped training the networks when the validation set error stopped decreasing apart from the fluctuations due to the inherent stochastic noise.

Similarly, we varied the number of local regions of VQPCA-Eucl and VQPCA-Recon from 5 to 50 in increments of 5. We varied the number of local regions at each stage for VQPCA-E-MS and VQPCA-R-MS from 5 to 50 in increments of 5 for a two level

multi-stage quantizer. We varied the branching factor of VQPCA-E-T and VQPCA-R-T for two and three level tree structured quantizers from 2 to 9. We did not train some of these configurations, which resulted in prohibitive memory and/or computation time requirements. For example, for the image data described later in the chapter, training a three level tree structured quantizer with a branching factor of 10 for a 50 to 5 dimension reduction required around 50 megabytes memory.

In the next subsection, we discuss the results of our experiments with speech vowel data. We then discuss the results of experiments with image data.

5.1.2 Dimension reduction of speech

In our first set of experiments, we performed a dimension reduction of the discrete Fourier transform (DFT) coefficients of speech data. We used examples of the twelve monothongal vowels extracted from continuous speech drawn from the TIMIT database (Fisher & Doddington 1986). Each input vector consists of the lowest 32 DFT coefficients (spanning the frequency range 0-4kHz), time-averaged over the central third of the utterance. We partitioned the data set into a training set containing 1200 vectors, a validation set containing 408 vectors and a test set containing 408 vectors. The validation set was used for architecture selection as described earlier. The test set utterances are from speakers *not* represented in the training set or the validation set. Motivated by the desire to capture formant structure in the vowel encodings, we reduced the data from 32 to 2 dimensions.

We trained each algorithm with four different random initializations of the free parameters. Figure 5.1 shows the mean test set reconstruction errors (5.2) averaged across the random initializations, with 2σ error bars symmetric about the mean. Here σ is the standard deviation of the errors across the different initializations. Figure 5.2 shows the mean training times with 2σ limits, and Table 5.1 gives the reconstruction errors and the training times numerically.

In general, the VQPCA encodings have about half the reconstruction error of global PCA or five layer nets. The FLNs failed to obtain a significantly lower error than a global

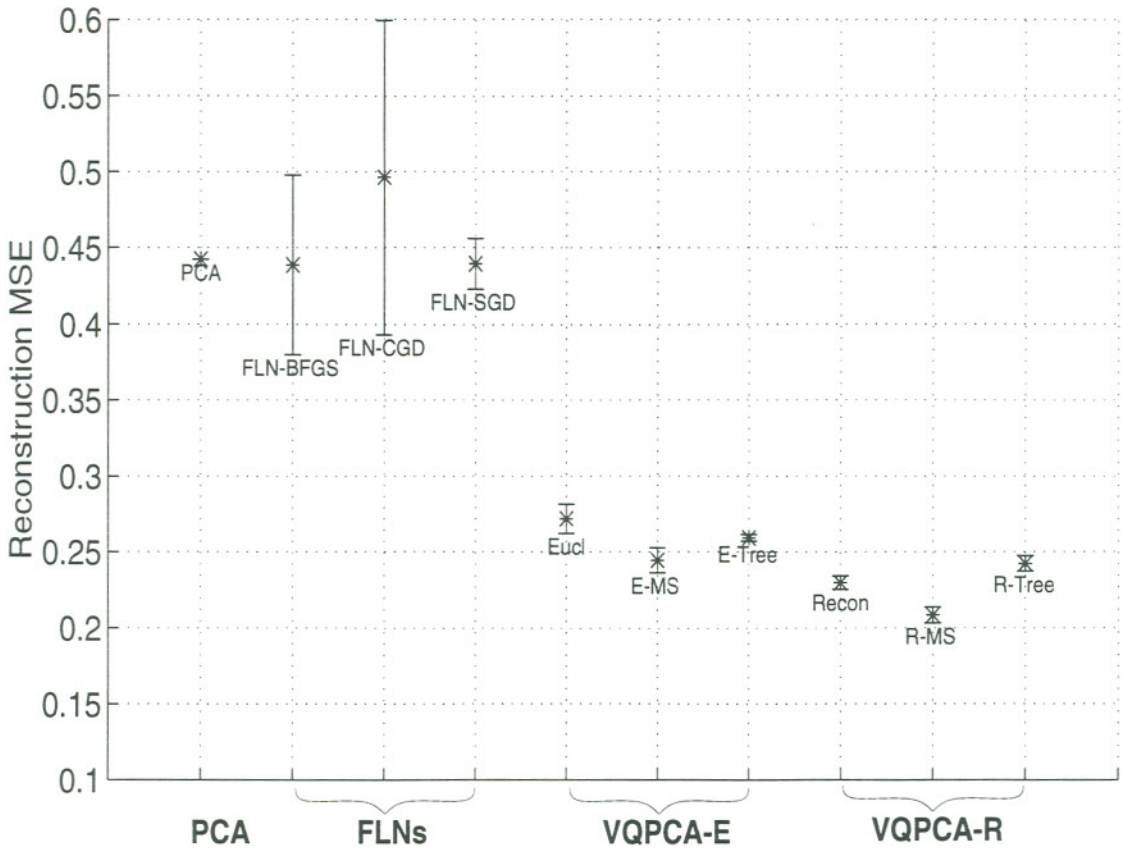

Figure 5.1: The test set reconstruction errors (5.2) for a 32 to 2 dimension reduction of TIMIT speech vowels (see text for details). The errorbars shown denote 2σ limits on either side of the mean for four different random initializations of the parameters for each algorithm. See Table 5.1 for the numerical values of the reconstruction errors plotted here.

PCA[1]. The FLNs seem to have a high variance in the reconstruction errors with respect to different random initializations of the weights. In fact, several nets have a higher error than PCA, indicating that these nets trapped into poor local optima. In contrast, all the local linear algorithms seem to be relatively insensitive to different random initializations of the training parameters.

[1]There is insufficient evidence to reject a null hypothesis that the mean reconstruction error for any of the FLN algorithms is the same as the PCA error using the students t test.

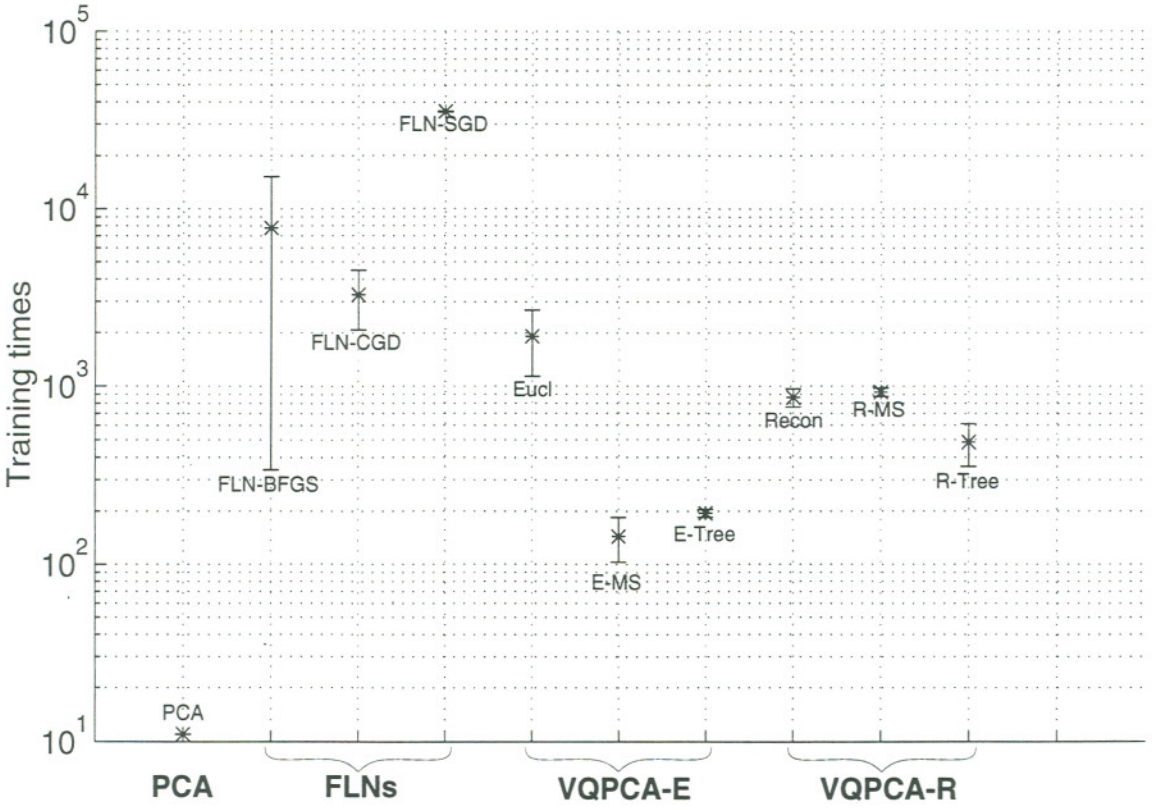

Figure 5.2: A plot of the time taken in seconds to train algorithms (on Sun Sparc 2 workstations) to reduce the dimension of TIMIT speech vowels from 32 spectral features to 2 dimensions. The errorbars shown denote 2σ limits on either side of the mean for four different random initializations of the parameters for each algorithm. The range of the variation in the training times was between $2622 - 3843$ for FLN-CGD and between $4,536 - 12,964$ for FLN-BFGS. The errorbars can be misleading in these two cases since they indicate a possible training time which is lower than the range of variation.

Table 5.1: Speech data **test set** reconstruction errors (5.2) and training times. The training times are reported in seconds to train on a Sun Sparc 2 workstation. The numbers in the parentheses are the values of the free parameters for the algorithm represented (e.g 5LN-CGD (5) indicates a network with 5 nodes in both the mapping (2nd and 4th) layers, while VQPCA-Eucl (50) indicates a clustering into 50 Voronoi cells). The errorbars shown represent 2σ limits symmetric about the mean (2σ away from the mean on either side) for four different random initializations of the parameters. Note that the errorbars for the training times can be misleading for FLN-CGD and FLN-BFGS since the actual range of variation is between $2,622 - 3,842$ for FLN-CGD and between $4,536 - 12,964$ for FLN-BFGS.

ALGORITHM	$\mathcal{E}_{\mathbf{norm}}$	**TRAINING TIME** (seconds)
PCA	0.443	11
FLN-CGD (35)	0.496 ± .103	7,784 ± 7442
FLN-BFGS (20)	0.439 ± .059	3,284 ± 1206
FLN-SGD (35)	0.440 ± .016	35,502 ± 182
VQPCA-Eucl (50)	0.272 ± .010	1,915 ± 780
VQPCA-E-MS (40x40)	0.244 ± .008	144 ± 41
VQPCA-E-T (15x15)	0.259 ± .002	195 ± 10
VQPCA-Recon (45)	0.230 ± .004	864 ± 102
VQPCA-R-MS (45x45)	0.208 ± .005	924 ± 50
VQPCA-R-T (9x9)	0.242 ± .005	484 ± 128

From Figure 5.1, we note that clustering with reconstruction distance produced encodings with a lower error than those produced by clustering with Euclidean distance. We obtained the most accurate encoding (least error) using VQPCA-R-MS. Both the multi-stage algorithms obtained a lower error than the corresponding flat quantizer algorithms. This may be because of the regularizing effect of the constraints imposed on the quantization when using a multi-stage VQ. A two level multi-stage VQ with Q cells at each stage effectively quantizes the input space into Q^2 cells (see section 3.1.4), but the shape of the Q^2 cells is constrained or restricted. This regularizing effect can be helpful when training with a large number of parameters.

From Figure 5.2, we note that FLNs are very slow to train. VQPCA algorithms with a multi-stage or tree structured architecture (except for VQPCA-R-MS) train more than

Table 5.2: The encode and decode times for different algorithms for a 32 to 2 dimension reduction of TIMIT speech vowels.

ALGORITHM	ENCODE TIME (FLOPS)	DECODE TIME (FLOPS)
PCA	158	128
5LN-CGD (35)	2,380	2,380
5LN-BFGS (20)	1,360	1,360
5LN-SGD (35)	2,380	2,380
VQPCA-Eucl (50)	4,957	128
VQPCA-E-MS (40x40)	7,836	192
VQPCA-E-T (15x15)	3,036	128
VQPCA-Recon (45)	87,939	128
VQPCA-R-MS (45x45)	96,578	192
VQPCA-R-T (9x9)	35,320	128

an order of magnitude faster (sometimes more than two orders of magnitude faster) than the FLNs, and achieve nearly half the error incurred by the FLNs. VQPCA-Eucl, which uses competitive learning is slower than VQPCA-Recon, which uses a batch-mode GLA. However, both are much faster than most of the FLNs.

Table 5.2 shows the encode and decode times for different algorithms. We note that the VQPCA algorithms (especially those using reconstruction distance clustering) require many more FLOPS to encode an input vector than five layer networks. However the decoding is much faster and comparable to PCA. The results indicate that VQPCA encoding may not be suitable for some real time applications (like video conferencing) where very fast encoding is desired. However, when only the decoding speed is of concern (e.g multi-media image retrieval), VQPCA algorithms are desirable because of the accuracy of their encodings, small training times and fast decoding.

In order to test whether the results described above were specific to a particular training data set, we shuffled all the data and repartitioned it into new training, validation and test sets of the same size as above. Figure 5.3 summarizes the results of experiments using the new data sets. Note the close correspondence of all the numbers between Figure 5.1 and Figure 5.3. This indicates that the results discussed in in the

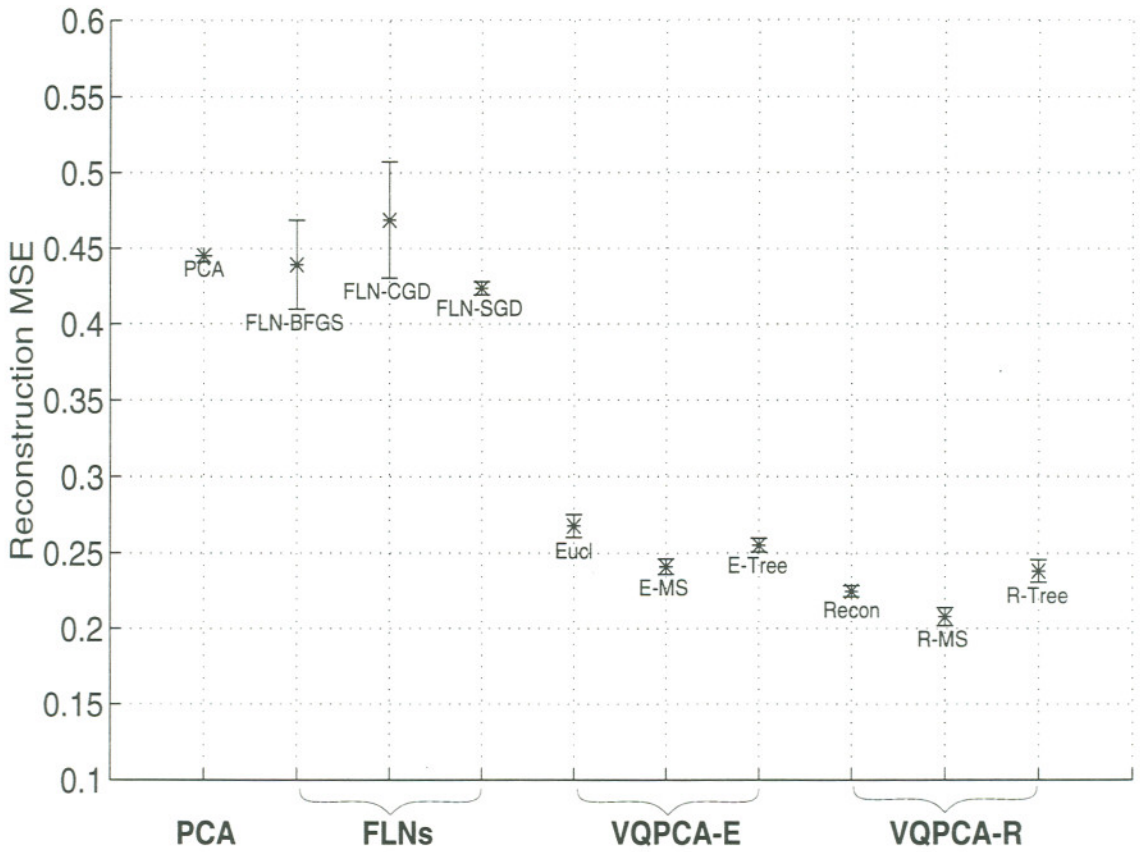

Figure 5.3: Plot showing the normalized test set reconstruction errors (5.2) of TIMIT speech vowels for a different assignment of the data into training, validation and test sets. The plot shows the average test set errors across four different random initializations of the parameters for each algorithm along with 2σ error bars symmetric about the mean. Note the close correspondence between this plot and the plot in Figure 5.1. This indicates that the results in Figure 5.1 were not specific to some training set.

preceding pages are not particular to a specific training set.

5.1.3 Dimension reduction of images

In our next set of experiments, we reduced the dimension of of grayscale image data. The data consists of 160 images of the faces of 20 people. Each person was asked to feign 8 different emotional states (astonished, happy, pleased, relaxed, sleepy, bored, miserable and angry). Each image is a 64x64, 8-bit/pixel grayscale image. We obtained the data from Gary Cottrell and David DeMers at UCSD (Cottrell & Metcalfe 1991, DeMers & Cottrell 1993). Cottrell and Metcalf (1991) generated this data and used it to classify the identity, gender and emotions of humans using a neural network classifier.

We consider each image to be a point in a 4096 dimensional *face space*. We extracted the first 50 principal components of each image and use these as our experimental data. This is the same data and preparation that DeMers and Cottrell used in their study of dimension reduction with five layer auto-associative nets (DeMers & Cottrell 1993). They trained auto-associators to reduce the 50 principal components to 5 dimensions.

We divided the data into a training set containing 120 images, a validation set (for architecture selection) containing 20 images and a test set containing 20 images. We reduced the images to 5 dimensions using PCA, FLNs[2] and VQPCA. Each algorithm was separately trained with four different random initializations of the free parameters. Figure 5.4 shows the mean test set reconstruction errors (5.2) averaged across the random initializations, along with 2σ error bars symmetric about the mean, where σ is the standard deviation of the errors across the different initializations. Figure 5.5 shows the mean training times with 2σ limits and Table 5.3 gives the reconstruction errors and the training times numerically.

We note from Figure 5.4 that the five layer networks obtain about a 30% lower error than the PCA. Among the FLNs, FLN-SGD obtained the lowest error. Both VQPCA-Eucl and VQPCA-Recon obtain a 40% lower error than FLN-SGD. Though on average,

[2]We tried FLNs with a configuration of $50 - n - 5 - n - 50$, n varying from 5 to 40 in increments of 5. The memory requirements were prohibitive for $n > 40$.

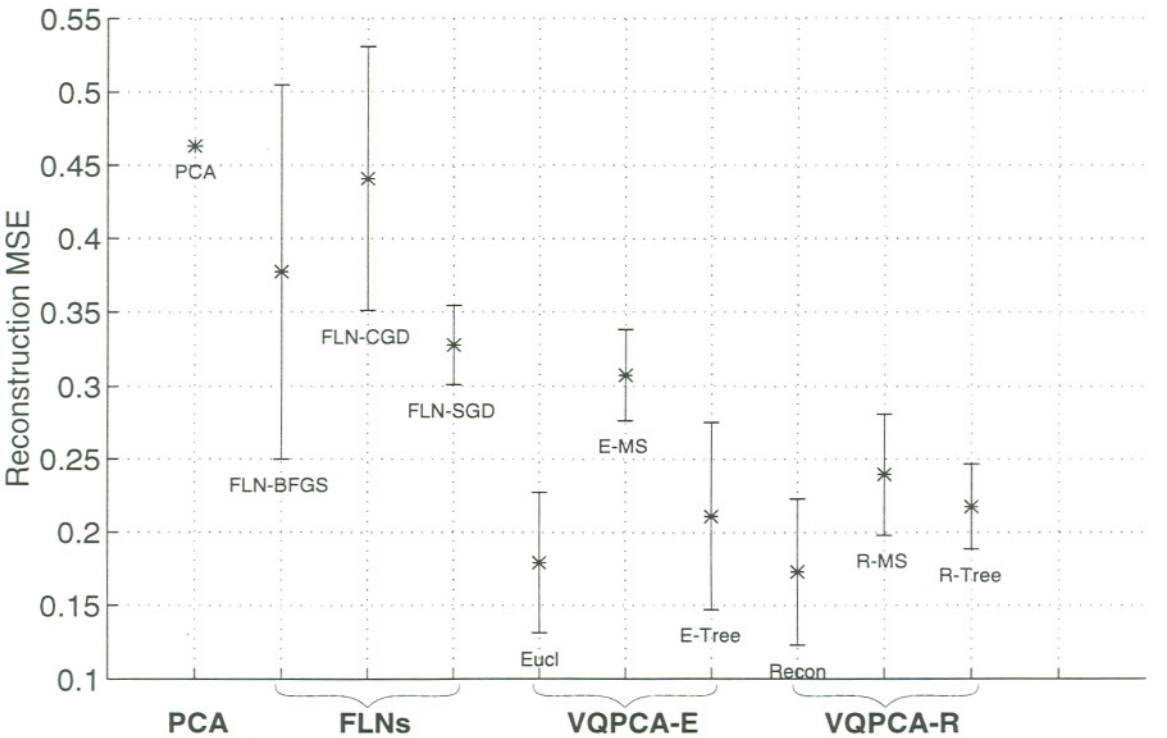

Figure 5.4: This figure plots the test set reconstruction errors (5.2) for a 50 to 5 dimension reduction of the faces data (see text for details). Each algorithm was trained with four different random initializations of the parameters. The plot shows the mean of the four trials along with 2σ limits on either side of the mean. See Table 5.3 for the numerical values of the reconstruction errors plotted here.

VQPCA-Recon obtained a lower error than VQPCA-Eucl, there is not much difference in the errors obtained by VQPCA-Eucl and VQPCA-Recon. The multi-stage and tree structured quantizers obtained lower errors than FLNs, but higher than VQPCA-Eucl or VQPCA-Recon.

Comparing the training times of different algorithms in Figure 5.5, we note that the local linear techniques take a significantly shorter time to train than the FLNs. FLN-CGD trains fast relative to the other network optimization schemes[3] but seems to

[3]We were unable to train the FLN-BFGS (35) reported in Table 5.3 on a Sun Sparc 2 because of prohibitive memory requirements. We trained the network on a Sparc 10 and converted the recorded

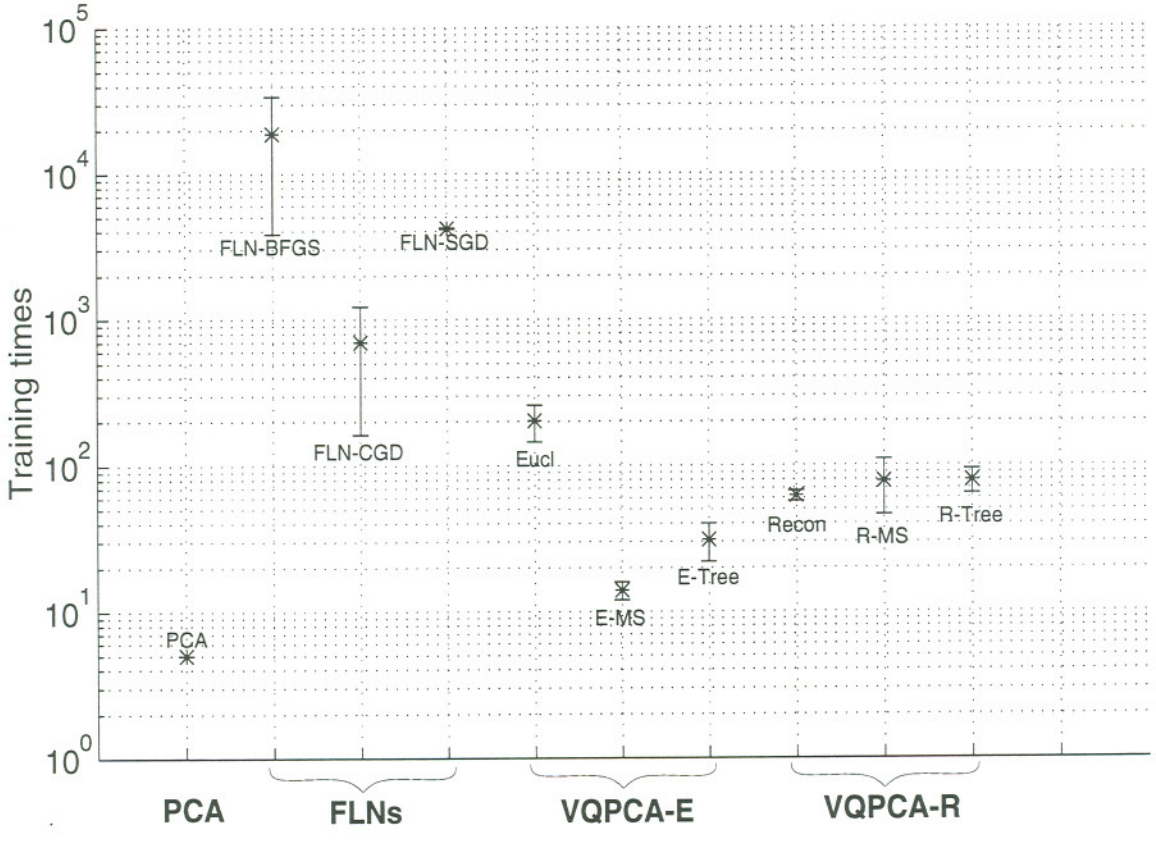

Figure 5.5: The training times, in seconds (on a Sun Sparc 2) for different algorithms for a 50 to 5 dimension reduction of the faces data. See Table 5.3 for numerical values of the training times plotted here.

Table 5.3: Reconstruction errors and training times for a 50 to 5 dimension reduction of the faces data. We report only those architectures which obtained the least validation set error over the parameter ranges explored. The training times are reported in seconds for training on Sun Sparc 2 workstations. Note that the errorbars for the training times can be misleading for FLN-CGD and FLN-BFGS, since the actual range of variation of training times is between $593 - 1,093$ for FLN-CGD and between $8,452 - 25,031$ for FLN-BFGS.

ALGORITHM	$\mathcal{E}_{\mathbf{norm}}$	**TRAINING TIME** (seconds)
PCA	0.463	5
5LN-CGD (35)	0.441 ± .090	698 ±533
5LN-BFGS (35)	0.377 ± .127	18,905 ±15,081
5LN-SGD (25)	0.327 ± .027	4,171 ±41
VQPCA-Eucl (20)	0.179 ± .048	202 ±57
VQPCA-E-MS (5x5)	0.307 ± .031	14 ±2
VQPCA-E-Tree (4x4)	0.211 ± .064	31 ±9
VQPCA-Recon (20)	0.173 ± .050	62 ±5
VQPCA-R-MS (20x20)	0.240 ± .042	78 ±32
VQPCA-R-Tree (5x5)	0.218 ± .029	79 ±15

generate inferior encodings (with respect to reconstruction errors). We again note that VQPCA-Eucl which uses a stochastic parameter update takes a significantly longer time to train than VQPCA-Recon (which uses a batch-mode GLA) with the same number of cells. Both VQPCA-Recon and VQPCA-Eucl are more than an order of magnitude faster than the best FLNs.

Table 5.4 shows the encode and decode times for different algorithms in FLOPS. We again note that VQPCA algorithms using reconstruction distance clustering (VQPCA-Recon, VQPCA-R-MS and VQPCA-R-T) use a much larger number of FLOPS to encode an input vector than FLNs or VQPCA algorithms using Euclidean distance clustering. However, the decode time is much smaller and comparable to other algorithms. As we noted earlier, this may indicate that these algorithms are not applicable to real time data processing applications like video conferencing, etc. However, when the encoding

time to the (roughly) corresponding time on a Sparc 2.

Table 5.4: The encode and decode times (in FLOPS) for a 50 to 5 dimension reduction of the faces data.

ALGORITHM	ENCODE TIME (FLOPS)	DECODE TIME (FLOPS)
PCA	545	500
5LN-CGD (35)	3,850	3,850
5LN-BFGS (35)	3,850	3,850
5LN-SGD (25)	2,750	2,750
VQPCA-Eucl (20)	3,544	500
VQPCA-E-MS (5x5)	2,043	600
VQPCA-E-T (4x4)	1,743	500
VQPCA-Recon (20)	91,494	500
VQPCA-R-MS (20x20)	97,493	600
VQPCA-R-T (5x5)	46,093	500

can be slow and only real time decoding is desired, VQPCA algorithms are preferable, particularly the ones using reconstruction distance clustering.

To test the sensitivity of the results with respect to the specific data set used, we collected all the data, shuffled it, and repartitioned it into training, validation and test sets of the same size as above. We trained all the algorithms using the new data. The resulting test set (the new test set) reconstruction errors are shown in Figure 5.6. Note the close correspondence between the errors reported in Figure 5.4 and those in Figure 5.6. This indicates that the results discussed above are not particular to some specific data set used.

For comparison with DeMers and Cottrell's (1993) work, we also conducted experiments training with *all* the data. The results are summarized[4] in Table 5.5. Both non-linear techniques (FLNs and VQPCA) produce encodings with lower error than PCA, indicating significant non-linear structure in the data. With the same data, and with a FLN with 30 nodes in each mapping layer, DeMers (1993) obtains a reconstruction

[4]For FLNs, we only show results with SGD in order to compare with the experimental results of DeMers. For this data, FLN-CGD produced encodings with a higher error and FLN-BFGS posed prohibitive memory and computational requirements.

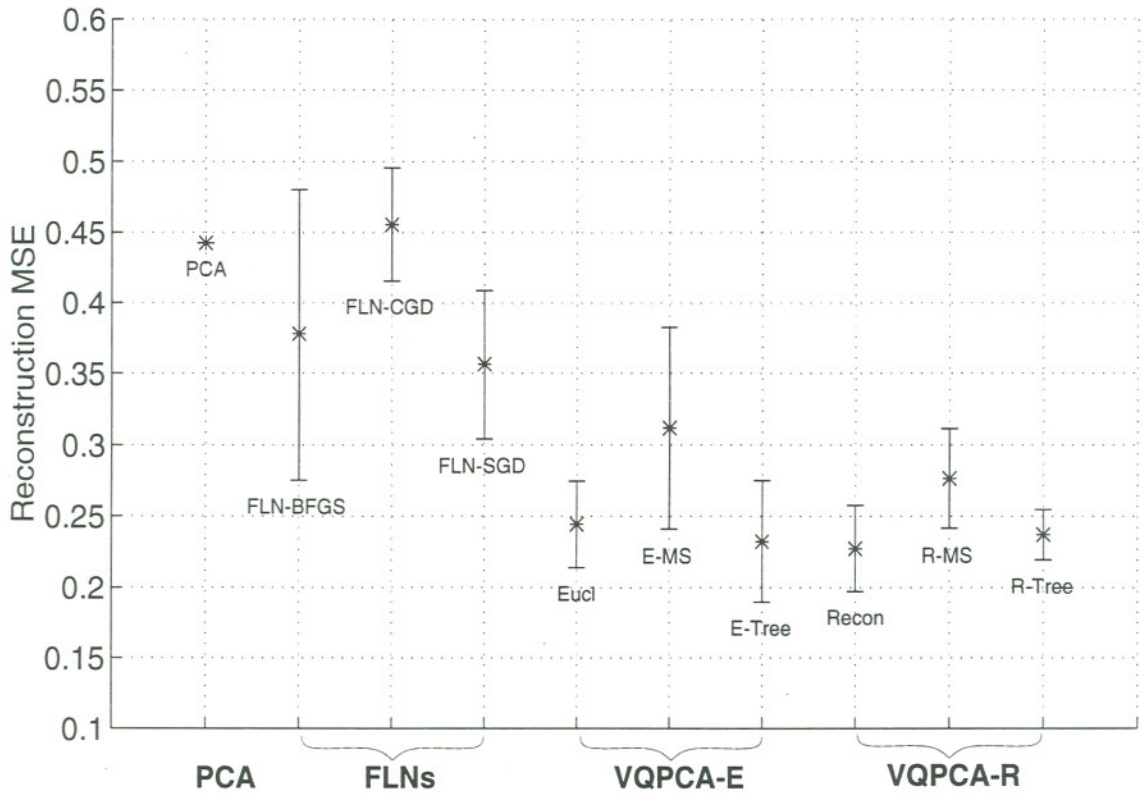

Figure 5.6: Plot showing the normalized test set reconstruction errors (5.2) with 2σ limits for the image data for a different assignment of the data into training, validation and test sets. Note the similarity between Figure 5.4 and the figure above.

Table 5.5: Reconstruction errors and training times for a 50 to 5 dimension reduction of images (training with *all* the data). The training times are reported in seconds to train on a Sun Sparc 2 workstation. We report architectures which obtained the least error over the parameter ranges explored.

ALGORITHM	$\mathcal{E}_{\mathbf{norm}}$	**TRAINING TIME** (in seconds)
PCA	0.405	7
5LN-SGD (30)	0.103	25,306
5LN-SGD (40)	0.073	31,980
VQPCA-Eucl (25)	0.026	251
VQPCA-Recon (30)	0.022	116

Figure 5.7: Two representative images: Left to right – Original 50 principal components reconstructed image, reconstruction from 5-D encodings: PCA, 5LN-SGD(40), VQPCA-Eucl(25), and VQPCA-Recon(30). The normalized reconstruction errors and training times for the whole data set (all the images) is given in Table 5.5.

error $\mathcal{E}_{norm}$ 0.1317[5]. We note that the VQPCA algorithms achieve an order of magnitude improvement over five layer nets in terms of speed of training while obtaining more accurate encodings.

Figure 5.7 shows two sample images from the data set and their reconstructions from 5 dimensional encodings using different algorithms. The algorithms represented are the same ones reported in Table 5.5. Looking at Figure 5.7, it is clear that the PCA encoding is inadequate, since it is hard to (perceptually) recognize gender or identity

[5]DeMers reports half the MSE per output node, $E = (1/2) * (1/50) * MSE = 0.001$. This corresponds to $\mathcal{E}_{norm} = 0.1317$

from the reconstructed image. Both the FLN and the VQPCA algorithms produce very (perceptually) good quality reconstructions. The difference between the reconstructions produced by the FLN and the VQPCA algorithms seems to be minor. Thus, we have been able to encode 4096 dimensional inputs (image pixels) using 5 real numbers with very little perceptually noticable degradation!

5.1.4 Summary of dimension reduction experiments

Our results with dimension reduction experiments were as follows.

- VQPCA algorithms obtained about half the error (40%-50%) obtained by global PCA or FLNs.
- VQPCA algorithms trained nearly an order of magnitude faster (sometimes more than two orders of magnitude faster) than the best FLN.
- Several FLNs trapped into poor local optima, generating encodings with a higher error than PCA encodings.
- For speech data, clustering with reconstruction distance (VQPCA-Recon) produced a lower error than clustering with Euclidean distance (VQPCA-Eucl). There was not much difference for image data which had much fewer data points.
- VQPCA algorithms require more floating points operations (FLOPS) to encode a vector than FLNs, in particular, the algorithms using reconstruction distance clustering.

We conclude that the local linear algorithms can generate *more accurate* encodings than FLNs while training much *faster* than FLNs. In the next section, we describe an application of the dimension reduction algorithms for speech feature extraction. We show the low dimensional encodings of speech vowels obtained by PCA, FLNs and VQPCA are capturing the formant frequency information in an encoded form.

5.2 Experiments with speech feature extraction

In this section, we build three dimensional encodings of spectral coefficients of speech vowels using dimension reduction algorithms. The goal is to study whether the low dimensional representations are able to capture the *formants* information in some encoded form.

Formants are the resonant frequencies of the vocal tract. As the speech articulators such as the tongue move towards different target positions on the roof of the mouth, the corresponding formant frequencies change and vary over a wide range. The formants appear as characteristic peaks in vowel spectra. We are particularly interested in the first three formants (F1, F2 and F3) since they provide the most discriminatory information about vowels (e.g see (Pols, Tromp & Plomp 1973)).

Peterson and Barney (1952) showed that vowel categories are well separated by formant frequencies in a perceptual study. Pols, Tromp and Plomp (1973) performed a statistical analysis of hand-labelled formant frequencies and formant levels of 12 Dutch vowels and concluded that "F1 and F2 are the most appropriate two distinctive parameters for describing the spectral differences among the vowel sounds". D. O'Shaughnessy showed (1987) that the lowest three formants are sufficient for producing intelligible speech.

Plomp *et al* (1967) performed a 4 dimensional PCA of 18 dimensional feature vectors (sound pressure levels in 18 frequency bands) of 15 Dutch vowels and noted similarities between scatter plots of the first two principal components and a scatter plot of the first two hand-labelled formant frequencies plotted by Peterson and Barney (1952). Pols *et al* (1969) report finding strong correlations between the principal components of spectral features and coefficients derived from a perceptual study for discriminating 11 Dutch vowels.

In this section, we first build three dimensional encodings of the raw spectral data (lowest 32 DFT coefficients) using PCA, FLNs and VQPCA. We then build a linear map from the encodings to hand-labelled formant frequencies.

5.2.1 Experimental setup

For the formant mapping experiments, we used vowels extracted from spoken letters. The data is from isolated utterances of the letters A,E,F,O and R by females only from the ISOLET database (Cole, Muthusamy & Fanty 1990). Utterances of A,E,F,O and R contain the phonemes, /ey/,/iy/,/eh/,/ow/ and /aa/ respectively. The input data consists of the lowest 32 DFT coefficients (from 0 to 4KHz frequency range), time-averaged over the central third of the vowel. Each utterance of a letter corresponds to one pattern in the training set. We partitioned the data into a training set containing 225 patterns, a validation set containing 75 patterns, and a test set containing 75 patterns. We hand-labelled the formant frequencies of both the test sets. We reduced the data from 32 to 3 dimensions using PCA, FLNs and VQPCA. We then built linear maps using linear ridge regression (see section 7.1) from the low dimensional encodings to the hand-labelled formant frequencies. This is similar to the approach described by Broad and Clermont (1989) who present a method of estimating formant frequencies from linear combinations of cepstral coefficients.

Dimension reduction algorithm	$\mathcal{E}_{norm}$
PCA	0.137
FLN-BFGS (32-25-3-25-32)	0.109
VQPCA-Recon (15 cells)	0.084

Table 5.6: Results of dimension reduction experiments with the ISOLET speech vowel data:: This table shows the normalized mean squared reconstruction errors (5.2) for the test set for the dimension reduction of the ISOLET vowel data (see text for details) from 32 to 3 dimensions using different algorithms.

We trained PCA, FLNs and VQPCA to reduce dimension of the data from 32 to 3 dimensions. We trained several FLN-BFGS networks varying the number of nodes in the mapping layers from 5 to 50 in increments of 5. We obtained the least validation set error for an FLN with a configuration of 32-25-3-25-32. We trained VQPCA-Recon varying the number of local regions from 5 to 50 in increments of 5. We obtained the

least validation set error with 15 local regions. Table 5.6 summarizes the normalized test set reconstruction errors (5.2) obtained for PCA, FLNs and VQPCA. From Table 5.6, we see that both FLN and VQPCA produced more accurate (lower error) encodings than PCA and that VQPCA has a lower error than the FLN.

5.2.2 Formant estimation

In the next phase of our experiment, we built a linear map from the low dimensional encodings obtained above to hand-labelled formant frequencies. A mapping is necessary because the low dimensional encodings may represent a function of the formant frequencies rather than F1, F2 and F3 directly. We measure the the "closeness" of the low dimensional encodings to formant frequencies by the average squared error incurred by the linear maps from the encodings to the hand-labelled frequencies[6].

For building the linear map, we hand-labelled the first three formants of each vowel in our data set. We used the low dimensional encodings of the validation set as the training set for building the mapping, and we used the low dimensional encodings of the test set as the test set for the mappings. Suppose f_i and $\hat{f}_i$ are the vectors of hand-labelled and estimated formant frequencies for the i^{th} input pattern. We measure the quality of formant estimation by the normalized estimation mean square error,

$$\mathcal{E}_{estimate} = \frac{\frac{1}{N}\sum_{i=1}^{N} \|f_i - \hat{f}_i\|^2}{\frac{1}{N}\sum_{i=1}^{N} \|f_i - \bar{f}\|^2} , \tag{5.3}$$

where $\bar{f} \equiv \frac{1}{N}\sum_{i=1}^{N} f_i$, and N is the number of patterns.

We used linear ridge regression to obtain the linear mappings. For VQPCA, we used an input representation consisting of a unary representation of the index of the local region (e.g cell number 2 out of 4 cells corresponds to $0.0, 1.0, 0.0, 0.0$) followed by the local three dimensional coordinates. Note that a linear map not be able to extract the locality information from the VQPCA encoding and hence may generate a higher

[6]We also trained non-linear maps using feedforward neural networks from the low dimensional encodings to hand-labelled formant frequencies. We obtained the same ranking of algorithms in terms of the formants estimation error as using linear maps.

error. For comparison, we also built a linear map from the 32 raw spectral coefficients to hand-labelled formant frequencies.

Dimension reduction algorithm	$\mathcal{E}_{estimate}$
PCA	0.171
FLN (32-25-3-25-32)	0.151
VQPCA (15 cells)	0.218
Direct map $32 \Rightarrow 3$	0.129

Table 5.7: Results of formant mapping experiments :: This table shows the normalized test set formant estimation errors (5.3) for a linear map from the low dimensional encodings obtained by different dimension reduction algorithms to hand-labelled formant frequencies. The last row represents a direct linear mapping from the 32 spectral coefficients to formant frequencies.

Table 5.7 shows the test set formant estimation errors (5.3) for the linear mappings discussed above. The direct map from the 32 spectral coefficients to the formant frequencies obtained the least error. All three dimension reduction algorithms (PCA, FLN and VQPCA) obtain a slightly higher error than the direct mapping. All algorithms obtain a normalized error $\mathcal{E}_{estimate}$ less than 0.25. We note that both the global techniques (PCA and the FLN) obtain a lower error than the local linear technique (VQPCA) and the FLN has a lower error than PCA. The VQPCA encoding of $x \in \mathcal{R}^{32}$ is $< w, z >$ where w is a representation of the index of local region and $z \in \mathcal{R}^3$ is the local coordinate vector. A linear map may be unable to extract the locality information from the VQPCA encoding. From the low estimation errors, we conclude that PCA, FLNs and VQPCA are all encoding the first three formants in some form.

Figure 5.8 shows scatter plots for the test set of hand-labelled formants and formants estimated by an FLN (configuration: $32-25-3-25-32$) dimension reduction followed by a linear mapping to the formant space. The normalized test set error was $\mathcal{E}_{estimate} = 0.151$ (as reported in Table 5.7). Note that the cluster of estimated formant frequencies seems to approximate the cluster of hand-labelled formant frequencies quiet well, except for the letter "O", for which the estimated frequencies are way off.

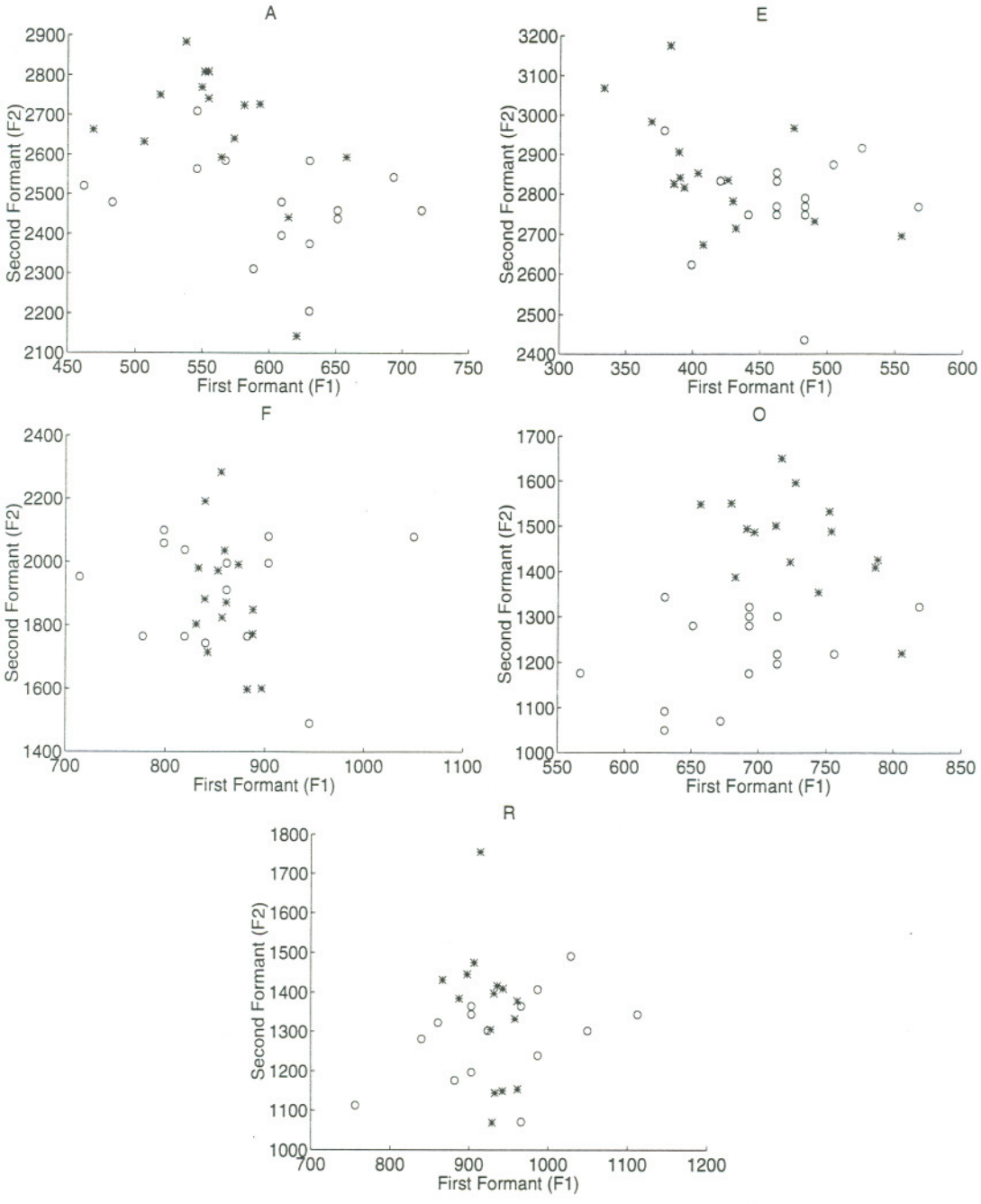

Figure 5.8: Overlaid scatter plots of F1 vs. F2 for hand-labelled formant frequencies (circles; o's) and the estimated formant frequencies (dots; *'s) for the test set. From top left, the plots are for the letters, *A*, *E*, *F*, *O* and *R* respectively. We used an FLN (32-25-3-15-32) to obtain 3-D encodings and a built a linear map from the 3-D encoding to the formant space.

We were successful in estimating the formant frequencies from 3 dimensional encodings of 32 raw spectral coefficients using PCA, FLN and VQPCA. We obtained a low normalized error ($\mathcal{E}_{estimate} < 0.25$) for all three algorithms. We conclude that PCA, FLNs and VQPCA were encoding the first three formants in some form. Thus, dimension reduction algorithms can be used for speech feature extraction.

Dimension reduction algorithms can also be used for lossy data compression. In the next section, we present rate-distortion curves comparing the compression performance of VQ and VQPCA for lossy speech and image compression.

5.3 Data compression experiments

In this section, we compare VQ and VQPCA for lossy compression of speech and image data. We are interested in the accuracy of encodings obtained by data compression algorithms *for a given bit-rate*. Thus, data compression differs from dimension reduction. Lossy data compression algorithms cannot recover the original data exactly from the compressed representation. The goal is to minimize the *distortion* for a given *bit-rate* of encoding. We define the bit-rate as the number of bits used to encode an n dimensional vector x and the distortion measure D as the expected squared error between x and its reconstruction $\hat{x}$, normalized by the data variance,

$$D = \frac{E\left[\,\|x - \hat{x}\|^2\,\right]}{E[\,\|x - \mu\|^2\,]}\,, \tag{5.4}$$

where $\mu \equiv E[x]$ is the mean of the data.

We compare the data compression performance of the algorithms by plotting the average distortion as a function of the bit-rate. These curves, known as the *rate-distortion* curves (Gersho & Gray 1992, Cover & Thomas 1991) or $R(D)$ curves, enable us to identify the best compression algorithm[7] for a specified bit-rate (or alternatively, for a specified distortion threshold).

[7] I am grateful to Professor Andrew Fraser (PSU) for pointing this out.

In the next subsection, we describe data compression using VQ. We then describe a data compression algorithm using local linear dimension reduction (VQPCA; section 3.2.2). We present experimental results comparing $R(D)$ curves of VQ and VQPCA for speech and image data. When there is ample training data, VQ seems to always provide a lower distortion than VQPCA. When there is a paucity of data, for some bit-rates, VQPCA obtains a lower distortion than VQ.

5.3.1 Data compression using VQ

Vector quantization (VQ (Gersho & Gray 1992); section 3.1) is a standard technique for lossy data compression. Recall that a VQ with Q cells approximates a data vector x ($x \in \mathcal{R}^n$) as one of Q prototype vectors called codebook vectors $\{\mu_1, \ldots, \mu_Q\}$. A VQ using Euclidean distance as its distortion measure approximates x as the codebook vector closest (Euclidean distance) to x,

$$\hat{x} = \mu_w \quad , \tag{5.5}$$

where

$$w = \underset{k=1}{\operatorname{argmin}} \ \|x - \mu_k\|^2 \quad . \tag{5.6}$$

We pre-compute the codebook vectors using a VQ training algorithm (e.g the GLA; see section 3.1). The index w (5.6) of the closest codebook vector is sufficient to obtain $\hat{x}$ (5.5), the VQ approximation of x. Thus, VQ reduces the storage for x from n real numbers to $\log_2(Q)$ bits needed to specify w. The bit-rate of compression is $R = \log_2(Q)$ and the distortion is

$$D = \frac{E\left[\min_{k=1}^{Q} \ \|x - \mu_k\|^2\right]}{E[\,\|x - \mu\|^2\,]} \quad , \tag{5.7}$$

where $\mu = E[x]$.

5.3.2 Data compression using VQPCA

We propose the following procedure for data compression.

- Reduce the dimension of x from n to m ($m < n$) using VQPCA (section 3.2.2). Let $z = f(x)$ denote the m dimensional encoding.
- Quantize z using a VQ as in section 5.3.1.

Recall that VQPCA (section 3.2.2) with Q cells builds a local linear model of the data and represents a vector $x \in \mathcal{R}^n$ as $< w, z >$, where w is the index of the local region and $z \in \mathcal{R}^m$ is a vector of the local coordinates. VQPCA approximates x as

$$x' = \mu_w + \sum_{i=1}^{m} e_{wi} e_{wi}^T (x - \mu_w) \equiv \mu_w + \sum_{i=1}^{m} z_i e_i \ , \tag{5.8}$$

where μ_w is the centroid for the w^{th} local region, e_{wi} are the orthonormal eigenvectors of the covariance matrix for the w^{th} region, z is a vector of the m local coordinates,

$$z = f(x) = (e_{w1}^T (x - \mu_w), \ldots, e_{wm}^T (x - \mu_w)) \ , \tag{5.9}$$

and w is the index of the local PCA hyperplane closest to x,

$$w = \underset{k=1}{\overset{Q}{\operatorname{argmin}}} \ (x - \mu_k)^T \left(\sum_{i=m+1}^{n} e_{ki} {e_{ki}}^T \right) (x - \mu_k) \ . \tag{5.10}$$

We propose quantizing the m dimensional vector z (5.9) using a VQ with Euclidean distance[8] with K codebook vectors $\{c_1, \ldots, c_K\}$, where $c_i \in \mathcal{R}^m$. This VQ approximates z with the codebook vector closest to z,

$$\hat{z} = c_d \ , \tag{5.11}$$

where

$$d = \underset{j=1}{\overset{K}{\operatorname{argmin}}} \ E \left[\| z - c_j \|^2 \right] \ . \tag{5.12}$$

We reconstruct x as

$$\hat{x} = \mu_w + \sum_{i=1}^{m} \hat{z}_i e_i \ , \tag{5.13}$$

using the quantized local encoding $\hat{z}$ instead of the local encoding z.

[8] An alternative is to scalar quantize each component of z separately as in transform coding (Gersho & Gray 1992). However, a VQ is more efficient and will induce smaller quantization error.

Quantizing the local encoding z enables us to compress the VQPCA representation of x. The index d (5.12) of the codebook vector closest (Euclidean distance) to z is sufficient to obtain $\hat{z}$ (5.11). The quantized VQPCA encoding is $< w, d >$, where w (5.10) is the index of the local region for VQPCA and d (5.12) is the index of the closest codebook vector to z. The bit cost of specifying $< w, d >$ is $\log_2(Q)$ bits for specifying w, plus $\log_2(K)$ bits for specifying d. Thus, the bit-rate of VQPCA compression is $R = \log_2(Q) + \log_2(K)$ and the distortion D is

$$D = \frac{E\left[\,\|x - \hat{x}\|^2\,\right]}{E[\,\|x - \mu\|^2\,]} \quad , \tag{5.14}$$

where $\hat{x}$ is defined in (5.13), and $\mu = E[x]$.

We train the model using a training set $\mathcal{X} = \{x^1, \ldots, x^N\}$, where $x^I \in \mathcal{R}^n$, as follows:

- We train a VQPCA with Q cells (section 3.2.2) to reduce elements of $\mathcal{X}$ from n to m dimensions. Separate the training vectors into Q sets $\{\mathcal{X}_1, \ldots, \mathcal{X}_Q\}$ using the VQPCA partition into Q local regions. Let $\{\mathcal{Z}_1, \ldots, \mathcal{Z}_Q\}$ denote the corresponding sets of m dimensional encodings (i.e let $\mathcal{Z}_i$ denote the encodings of elements of $\mathcal{X}_i$) obtained using (5.9).

- For the i^{th} local region of the VQPCA partition, we train a VQ with Euclidean distance with K cells (section 3.1) to quantize $\mathcal{Z}_i$. We do this for all $i = 1, \ldots, Q$.

After training is complete, we compress a vector $x \in \mathcal{R}^n$ as follows:

- We compute the VQPCA encoding $< w, z >$ using (5.10) and (5.9).

- We quantize the local coordinate vector z using (5.11) and (5.12).

- The compressed encoding consists of w, the index of the VQPCA local region and d, the index of the quantization cell for z.

- We reconstruct x from the compressed encoding using (5.11) and (5.13).

Note that for the algorithm sketched above, several combinations of values of Q and K will result in the same bit-rate of compression. If the rate R is specified to us, we can

choose to have more local regions for VQPCA dimension reduction (larger Q) or we can choose to quantize the local encoding z at a finer resolution (larger K).

VQPCA approximates a two level tree structured VQ (TSVQ; see section 3.1.5) with Q cells in the first level and K second level cells in each first level cell. If we retain all the principal components in each first level cell (instead of reducing dimension), VQPCA is equivalent to a two level TSVQ or a two level tree structured transform coding (Gersho & Gray 1992). Reducing the dimension induces additional distortion. Hence a VQPCA approximates a TSVQ, and the approximation is better when the data lies near an m dimensional manifold and the distortion induced by the dimension reduction is small.

We now compare VQ and VQPCA for compression of speech and image data. For VQ compression, we vary the number of cells Q over a range of values and compute the distortion D for each Q. We plot the $R(D)$ curve. For VQPCA, we vary the number of local regions Q over a range of values, and for each Q, we vary K over a range of values. We compute the distortion D for each combination of Q and K. Thus, we get a family of $R(D)$ curves for VQPCA.

5.3.3 Speech compression

Our first data set consists of spectral coefficients of speech vowels extracted from continuous speech utterances from the TIMIT database (Fisher & Doddington 1986). This is the same data as in section 5.1.2. Here, we partitioned the 32 dimensional data into a training set containing 1200 vectors and test set containing 816 vectors.

We trained VQs with the number of cells Q varying from 1 to 256. We trained VQPCA models for a 32 to 5 dimension reduction, with the number of local regions Q varying from 1 to 20, and the number of cells for quantizing the local encoding, K, varying from 1 to 20.

Figure 5.9 shows the $R(D)$ curves of VQ and 5-D VQPCA for the average test set performance, averaged across three different random initializations of the parameters of VQ and VQPCA. The curve for VQ is shown dotted with *'s. Each solid line represents a VQPCA with fixed Q and varying K. The lines represent $Q = 1, \ldots, 20$ from left to

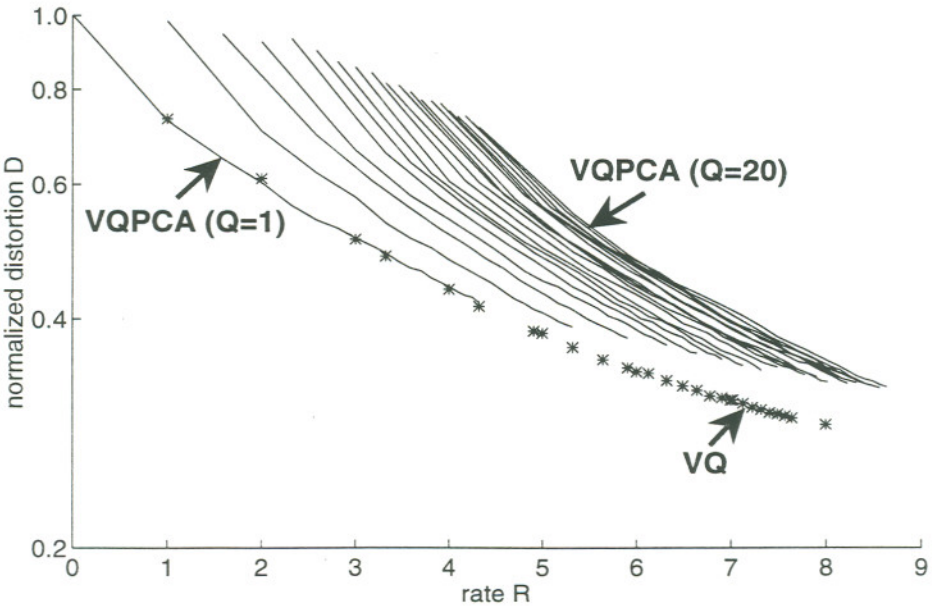

Figure 5.9: Rate distortion ($R(D)$) curves for TIMIT speech vowels for VQ (the curve dotted with *'s) and 5-D VQPCA (the set of lines) data compression. The figure plots the bit-rate on the horizontal axis and the mean normalized test set distortion on the vertical axis, averaged over three different random initializations of the parameters of the algorithms. The solid lines are the $R(D)$ curves of VQPCA with reconstruction distance clustering, with Q varying from 1 to 20, from left to right. Each line (VQPCA configuration) shows the decrease in distortion as we increase K from 1 to 20.

right.

From Figure 5.9, we note that

- for VQPCA, the $R(D)$ curve with $Q = 1$ obtains the least distortion at all bit-rates. Note that VQPCA with $Q = 1$ is just a global PCA. For a fixed R, as we increase Q, we are left with fewer bits (smaller K) for quantizing the local encodings. Thus, PCA (VQPCA with $Q = 1$) obtains a lower distortion since it can quantize the 5-D encodings at a finer resolution.

- The $R(D)$ curves of VQ and 5-D PCA (VQPCA with $Q = 1$) are almost identical. The former quantizes the 32 dimensional vector x, while the latter quantizes the 5 dimensional vector $z = f(x)$. The results suggest that most of the data points

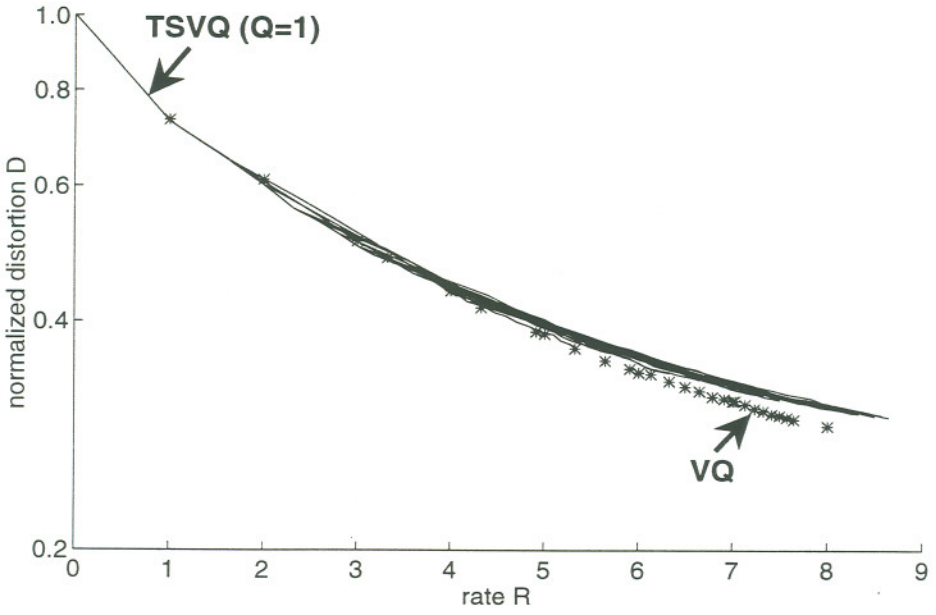

Figure 5.10: Rate distortion ($R(D)$) curves for TIMIT speech vowels for VQ (the curve dotted with *'s) and a TSVQ (the set of lines) with Q first level cells and K second level cells within each first level cells. We vary Q from 1 to 20 (the solid curves from left to right), and for each Q, we vary K from 1 to 20. The figure plots the bit-rate on the horizontal axis and the mean normalized test set distortion on the vertical axis, averaged over three different random initializations of the parameters of the algorithms.

lie near a 5-D subspace and hence quantizing $x \in \mathcal{R}^{32}$ is as effective as quantizing $z \in \mathcal{R}^5$. However, for large R, we expect the $R(D)$ curve for PCA to asymptote to the distortion induced by the dimension reduction.

Our results suggest that, for data compression using VQPCA with a specified R, we can lower the distortion by choosing a smaller Q (number of local regions for dimension reduction) and a larger K (number of codebook vectors for quantizing the local encoding). In other words, it is more advantageous to increase the precision of quantization of the local encodings than to increase the precision of dimension reduction. The extreme case is when $Q = 1$ (PCA),when all R bits are used for quantizing the encoding. Though the $R(D)$ curve of PCA approximates a VQ, it asymptotes to the distortion induced by the dimension reduction.

As discussed earlier, for data compression, VQPCA approximates a two level tree structured VQ (TSVQ; see section 3.1.5) with Q cells in the first level and K second level cells in each first level cell. Figure 5.10 shows a family of $R(D)$ curves, with Q varying from 1 to 20 (one curve for each Q) and for each Q, K varying from 1 to 20. Note that the TSVQ curves perform better than the VQPCA curves but worse than the VQ curve. For the same bit-rate, a TSVQ has a larger distortion than a VQ (see the discussion in section 3.1.5) because of the structural constraints imposed by TSVQ (second level cells are within first level cells; see section 3.1.5). Thus, since VQPCA approximates a TSVQ, for a given bit-rate, a VQ always obtains a lower distortion than VQPCA.

However, when the sample data size is limited, the structural constraints of a TSVQ or a VQPCA can have a regularizing effect, leading to a lower *test set* distortion than a VQ for the same bit-rate. In the next subsection, we present $R(D)$ curves for image data.

5.3.4 Image compression

Our image data consists of the leading 50 principal components of 64x64 8-bit grayscale images of the faces of people. This is the same data described in section 5.1.3. For $R(D)$ experiments, we partitioned the 50 dimensional data into a training set containing 120 vectors and a test set containing 40 vectors.

We trained several VQs with the number of cells Q varying from 1 to 120. We trained 5-D VQPCA with the number of local regions Q varying from 1 to 20, and the number of cells for quantizing the 5-D encoding, K, varying from 1 to 20. Figure 5.11 shows the $R(D)$ curves of VQ and 5-D VQPCA for the average test set performance, averaged across three different random initializations of the parameters of VQ and VQPCA. The curve for VQ is shown dotted with *'s. Each solid line represents a VQPCA with fixed Q and varying K. The lines represent $Q = 1, \ldots, 20$ from left to right.

We note that the $R(D)$ curves for VQPCA are much closer to the $R(D)$ curve for VQ than they were for speech data. In fact, for certain bit rates, the least distortion

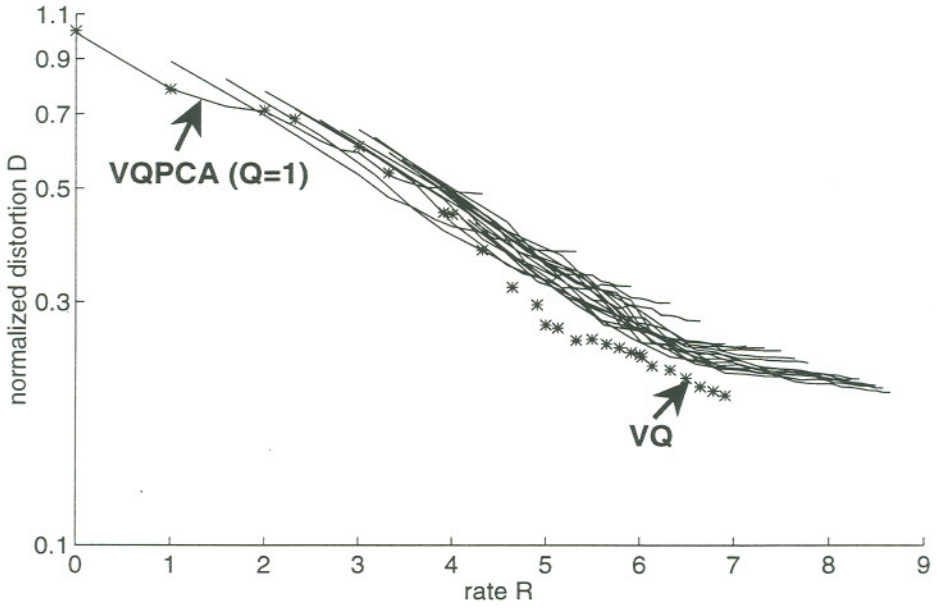

Figure 5.11: The rate distortion R(D) curves for the faces data for VQ (shown dotted with *'s) and 5-D VQPCA data compression. The horizontal axis is the rate R, and the vertical axis is the normalized mean test set distortion, averaged over three different random initializations of parameters.

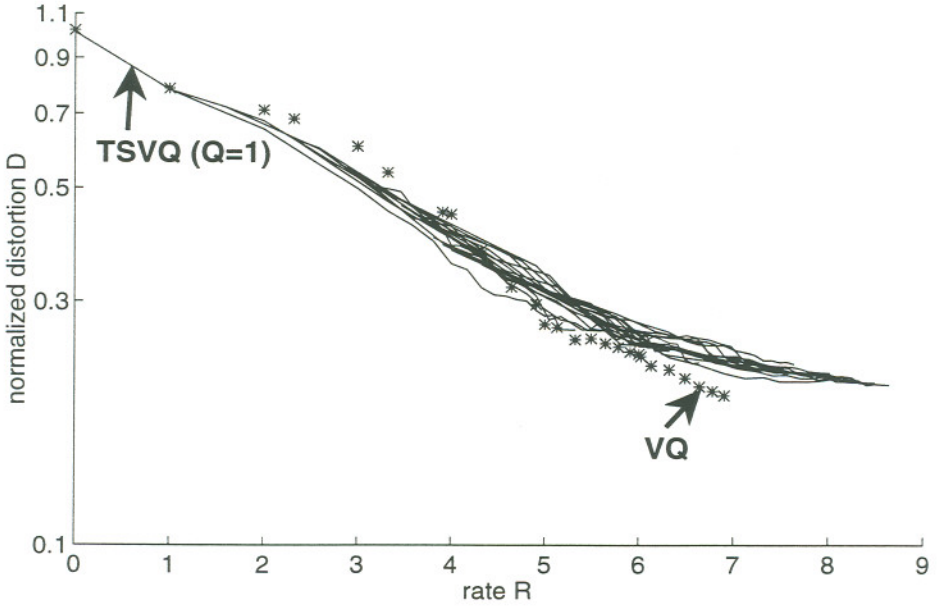

Figure 5.12: Rate distortion ($R(D)$) curves for the image data for VQ (the curve dotted with *'s) and a TSVQ (the set of lines) with Q first level cells and K second level cells within each first level cells. We vary Q from 1 to 20 (the solid curves from left to right), and for each Q, we vary K from 1 to 20. The figure plots the bit-rate on the horizontal axis and the mean normalized test set distortion on the vertical axis, averaged over three different random initializations of the parameters of the algorithms.

is obtained by VQPCA. This data set has very few training vectors. As mentioned above, the structural constraints of a TSVQ can have a regularizing effect, leading to a lower *test set* distortion than an unconstrained VQ for the same bit-rate. Since a VQPCA approximates a TSVQ and has similar structural constraints (the second level quantization is for subsets of the data), a VQPCA can have a lower distortion than VQ when the training sample size is limited. Figure 5.12 shows a plot of $R(D)$ curves for a TSVQ with Q cells in the first level and K sub-cells within each first level cell (we vary Q from 1 to 20, and for each Q, we vary K from 1 to 20). Note the similarity between the $R(D)$ curves for VQPCA and the corresponding curves for TSVQ.

5.3.5 Summary of rate-distortion experiments

In this section, we compared VQ and VQPCA for lossy compression of speech and image data. We compress data using VQPCA as follows:

- We reduce dimension of the data from n to $m < n$ using a VQPCA with Q cells.
- We quantize the m-D encodings using a VQ with K cells.

The compression has a bit-rate $R = \log_2(Q) + \log_2(K)$. Data compression using VQPCA approximates data compression using a two level tree structured VQ (TSVQ; section 3.1.5) with Q first level cells and K second level cells within each first level cells.

Based on our results with speech and image data, we conclude that

- In general, a VQ always obtains a lower distortion than VQPCA for the same rate. This follows from the fact that VQPCA approximates a TSVQ.
- For VQPCA data compression, for a given rate R, using a smaller number of local regions for dimension reduction and larger number of cells for quantizing the local encodings (smaller Q and larger R) seems to obtain a lower distortion.
- When we have limited data, a TSVQ or a VQPCA sometimes obtain a lower distortion than VQ for the same rate. This may be due to the structural constraints

of TSVQ and VQPCA, which may have a regularizing effect, thus lowering the *test set* distortion.

5.4 Discussion

In this chapter, we presented experimental results comparing global dimension reduction algorithms (PCA and FLNs) with our local linear algorithms (VQPCA) for dimension reduction of speech and image data and speech feature extraction. We compared VQ and VQPCA for lossy compression of speech and image data.

Based on our results with dimension reduction experiments we conclude that:

- The local linear (VQPCA) algorithms can produce more accurate encodings than FLNs or PCA while training much faster than FLNs.
- The VQPCA algorithms (especially the ones using reconstruction distance clustering) require a higher number of FLOPS than FLNs to encode a vector. However decoding is much faster and comparable to PCA. This suggests that VQPCA algorithms may not be the best choice for applications requiring real time encoding. However, for applications which can tolerate slow encoding and require real time decoding, VQPCA algorithms would be the method of choice because of their fast training, high accuracy and fast decoding.

Thus, local linear algorithms (VQPCA) can produce more accurate encodings than FLNs while training much faster than FLNs.

In section 5.2, we estimated the resonance frequencies of the vocal tract (formants) using 3 dimensional encodings of 32 raw spectral coefficients using PCA, FLNs and VQPCA. For all three algorithms, we obtained a low error suggesting that all the algorithms are encoding the first three formants in some form.

In section 5.3, we presented rate-distortion curves ($R(D)$ curves) comparing VQ and VQPCA for lossy compression of speech and image data. Data compression using VQPCA approximates data compression using a two level TSVQ. In general, a VQ always

obtains a lower distortion than VQPCA for the same rate. However, when trained with small data sets, VQPCA sometimes obtains a lower *test set* distortion than VQ for the same bit-rate. This may be because of the regularizing effect of the structural constraints of a VQPCA (analogous to the structural constraints of a TSVQ).

Thus, we have shown that local linear algorithms (VQPCA) perform very fast *and* accurate model fitting. They can be used for dimension reduction, feature extraction, exploratory data analysis and lossy data compression. In the next two chapters, we present local linear algorithms for supervised learning.

Chapter 6

Gaussian mixture models for classification

In this chapter, we present Gaussian mixture models for classification; *supervised* learning tasks where we predict the membership (output variable) of data points (input variables) to one of several classes or categories. In preceding chapters, we presented local linear algorithms for dimension reduction and showed their relation to a mixture of Gaussians data model. These algorithms are *unsupervised*, where the goal is to learn the statistical dependencies among the data variables, all variables are treated alike and learning is not forced by a teacher function. In contrast, supervised learning algorithms use "labelled" data to learn statistical dependencies between designated "input" and "output" variables.

In this chapter and in the next chapter, we present Gaussian mixture models for supervised learning based on similar ideas that lead to the probabilistic framework for dimension reduction algorithms (chapter 4). In this chapter, we apply a mixture of Gaussians model for classification tasks and present a new method of regularizing the mixture model. In the next chapter, we apply a mixture model to regression tasks to derive local linear algorithms and present regularization techniques for these algorithms.

Mixture models are widely used in applications where data can be viewed as arising from several populations mixed in varying proportions. Mixture models have been used as a clustering technique (Lazarsfeld & Henry 1968, Ganesalingam & McLachlan 1979, Basford & McLachlan 1985). Gaussian mixture models are used to tackle the "missing

data problem" (Dempster *et al.* 1977, Ghahramani 1994, Ghahramani & Jordan 1994*b*, Ghahramani & Jordan 1994*a*, Ormoneit & Tresp 1995) where the values of one or more of the data variables may be unavailable. Bregler and Omohundro (1994, 1995) use constrained Gaussian mixture models for interpolating between specified images in a image sequence. Jordan *et al* (1991, 1994). use mixture models for supervised learning in the "mixture of experts" architecture. Hinton *et al* (1995) use constrained Gaussian mixture models for handwritten character recognition. Ghahramani and Jordan (1994, 1994*b*, 1994*a*) use mixture models for density estimation for classification, regression and clustering. For a much more extensive discussion on mixture models and their applications, see the books by Everitt and Hand (1981), Titterington *et al* (1985) and McLachlan and Basford (1988).

We present Gaussian mixture Bayes (GMB) classifiers, which use a mixture of Gaussians model for each class-conditional density (we model the input variables *for each class separately* using a mixture model). Priebe and Marchette (1991) have described a classification algorithm which recursively fits a separate Gaussian mixture model for each class, where the number of components is allowed to grow with the data. Heck and Chou (1994) use a GMB model to classify machine failure modes. Ghahramani and Jordan (1994*a*, 1994*b*) and Ormoneit and Tresp (1995) have previously discussed GMB classifiers. Marroquin (1995) also describes GMB classifiers, which he calls "local Gaussian classifiers".

In this chapter, we study GMB classifiers, show their relation to clustering based classification algorithms and explore different ways of regularizing them. A mixture of Gaussians model soft partitions the data. We can obtain a hard partition by applying a *winner-take-all* assumption as discussed in chapter 1 and chapter 4. Probabilistic clustering using a mixture of Gaussians model is discussed in (Ganesalingam & McLachlan 1979, Duda & Hart 1973, Basford & McLachlan 1985, McLachlan & Basford 1988, Nowlan 1991). Nowlan (1991) showed that a winner-take-all mixture of Gaussians model with certain assumptions approximates a VQ clustering with Euclidean distance. Duda and Hart (1973) have previously suggested this relation. Using a similar approach,

we derive winner-take-all approximations to GMB classifiers and show their relation to hard clustering based classifiers like the learning vector quantization algorithm (LVQ; see appendix D).

Gaussian mixture classifiers and their winner-take-all approximations suffer from the curse of dimensionality (see chapter 1). A mixture model with Q Gaussians has $O(Q*n^2)$ parameters where n is the input dimensionality. In section 6.2, we discuss the following schemes for regularizing GMB classifiers;

- assuming that all covariance matrices are diagonal or spherically symmetric,
- bounding the covariance matrices by adding a constant diagonal matrix $\epsilon I_{n\times n}$ to each covariance matrix in each iteration of the EM algorithm (Ghahramani & Jordan 1994*a*, Ormoneit & Tresp 1995).
- We propose a new regularization scheme which prunes those directions of each covariance matrix, which induce the least bias when pruned.

In section 6.3, we compare the performance of the different mixture models and multi-layer perceptrons (MLPs) for two speech classification tasks. Our results indicate that

- GMB classifiers perform comparably to MLPs, and
- regularizing the mixture models is necessary to prevent singular covariance matrices and over-parametrization. In particular, bounding the covariance matrices while training the mixture models works best.

6.1 Gaussian Mixture Bayes (GMB) classifiers

A classifier assigns vectors from $\mathcal{R}^n$ (n dimensional feature space) to one of K classes, partitioning the feature space into K disjoint regions. The input vectors, called *feature vectors* or *observation vectors*, consist of sets of measurements that distinguish between the classes. Let x denote a feature vector ($x \in \mathcal{R}^n$), and $\{\Omega^I, I = 1, \ldots, K\}$ denote the classes. We denote the *a priori probability* of observing a feature vector from class Ω^I

as $p(\Omega^I)$. This probability reflects our prior knowledge of how likely we are to observe feature vectors from the class Ω^I. Let $p(x \mid \Omega^I)$ denote the *class-conditional probability density* function for x, i.e, the probability density function for x given that x belongs to class Ω^I. We denote the *a posteriori* probability of x belonging to class Ω^I as $p(\Omega^I \mid x)$.

For a given feature vector x of unknown class, the probability of error in class assignment is minimized by choosing the class Ω^L, where

$$\begin{aligned} L &= \underset{I=1}{\overset{K}{\operatorname{argmax}}}\ p(\Omega^I \mid x) \\ &= \underset{I=1}{\overset{K}{\operatorname{argmax}}}\ \frac{p(\Omega^I)\, p(x \mid \Omega^I)}{p(x)} \quad \text{using Bayes rule} \\ &= \underset{I=1}{\overset{K}{\operatorname{argmax}}}\ p(\Omega^I)\, p(x \mid \Omega^I) \ , \end{aligned} \tag{6.1}$$

since $p(x)$ is independent of I. The functions

$$\delta^I(x) = p(\Omega^I)\, p(x \mid \Omega^I) \tag{6.2}$$

are known as the discriminant functions and a feature vector x is assigned to class I if $\delta^I(x) > \delta^J(x)\ \ \forall J \neq I$. A *Bayes* classifier (Duda & Hart 1973) assigns classes to feature vectors based on a model of the class conditional densities, using the discriminant functions (6.2). Given the class conditional densities, this choice minimizes the classification error rate (Duda & Hart 1973).

We model each class conditional density, $p(x \mid \Omega^I)$, by a mixture composed of Q^I component Gaussians,

$$p(x \mid \Omega^I) = \sum_{j=1}^{Q^I} \frac{\alpha_j^I}{(2\pi)^{n/2}\sqrt{|\Sigma_j^I|}} \exp\left[-\frac{1}{2}(x-\mu_j^I)^T \Sigma_j^{I\,-1} (x-\mu_j^I)\right] \ ,$$

where μ_j^I and Σ_j^I are the mean and the covariance matrix of the j^{th} component of the mixture density for the I^{th} class. The Bayes discriminant functions using this model are

$$\hat{\delta}^I(x) = p(\Omega^I) \sum_{j=1}^{Q^I} \frac{\alpha_j^I}{(2\pi)^{n/2}\sqrt{|\Sigma_j^I|}} \exp\left[-\frac{1}{2}(x-\mu_j^I)^T \Sigma_j^{I\,-1} (x-\mu_j^I)\right] \ , \tag{6.3}$$

We will refer to a classifier using the above discriminant functions as a Gaussian mixture Bayes (GMB) classifier. Figure 6.1 shows an example of a class assignment using a GMB classifier for a two class problem.

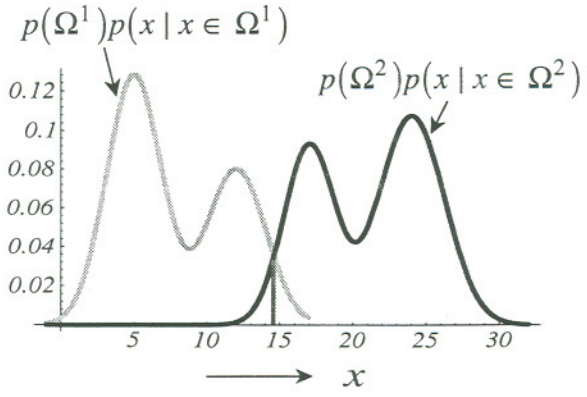

Figure 6.1: Figure showing the decision rule of a GMB classifier for a two class problem with one input feature. The horizontal axis represents the feature and the vertical axis represents the Bayes discriminant functions. In this example, the class conditional densities are modelled as a mixture of two Gaussians and equal priors are assumed.

To implement a GMB classifier we first separate the training data into the different classes. We then use the EM algorithm (Dempster *et al.* 1977, Nowlan 1991) to determine the parameters of the Gaussian mixture density for each class. The EM algorithm or the *Expectation Maximization* algorithm is widely used (Dempster *et al.* 1977, Nowlan 1991, Ghahramani 1994, Ghahramani & Jordan 1994*a*, Ghahramani & Jordan 1994*b*, Jordan & Jacobs 1994, Ormoneit & Tresp 1995) to iteratively generate maximum likelihood parameter estimates of mixture models. Appendix C describes an EM algorithm for estimating the parameters of a mixture of Gaussians model.

A mixture of Gaussians model builds a soft "local" model of the data, since each component Gaussian dominates the mixture density for some of the data points. In the next section, we derive winner-take-all GMB classifiers and show their relation to VQ clustering based classifiers. The winner-take-all classifiers partition the feature space *for each class separately* into disjoint regions using *only* the dominant Gaussian component for each data point.

6.1.1 The relation between GMB and VQ clustering based classifiers

We will now derive the relationship between GMB classifiers and VQ (see section 3.1) clustering based classifiers like the learning vector quantization (LVQ (Kohonen 1988): appendix D) algorithm. We will use a similar approach to Steven Nowlan's derivation showing the relation between Gaussian mixture models and unsupervised VQ clustering (see (Nowlan 1991) and appendix B).

The *winner-take-all* (WTA) assumptions relating VQ clustering to Gaussian mixtures (see (Nowlan 1991) and appendix B) are:

- $p(x \mid \Omega^I)$ are mixtures of Gaussians as in (6.3).
- The summation in (6.3) is dominated by the largest term.

These assumptions are "equivalent to assigning all of the responsibility for an observation to the Gaussian with the highest probability of generating that observation" (Nowlan 1991). We apply the above assumptions to the GMB discriminant functions (6.3). These assumptions imply that the discriminant functions effectively partition the input space *separately* for each class into disjoint regions.

To draw the relation between GMB and VQ clustering based classifiers, we further assume that the mixing proportions (α_j^I) are equal for a given class and the number of mixture components Q^I is proportional to $p(\Omega^I)$. Applying all of the above assumptions to (6.3), taking logs and discarding the terms that are identical for each class, we get the discriminant functions

$$\hat{\gamma^I}(x) \approx - \min_{j=1}^{Q^I} \left[\frac{1}{2} \log(|\Sigma_j^I|) + \frac{1}{2} (x - \mu_j^I)^T \, \Sigma_j^{I\,-1} \, (x - \mu_j^I) \right] . \tag{6.4}$$

If we further assume that for each class, the mixture components are spherically symmetric with covariance matrix $\Sigma_j^I = \sigma^2 I$, with σ^2 identical for all classes, we obtain the discriminant functions

$$\hat{\gamma^I}(x) \approx - \min_{j=1}^{Q^I} \| \, x - \mu_j^I \, \|^2 . \tag{6.5}$$

The discriminant functions in (6.5) are identical to those used by the LVQ algorithm ((Kohonen 1988); see appendix D). LVQ is a clustering based classification algorithm

which adapts the placement of reference vectors (or means) based on a set of class labelled data points. Though LVQ employs a discriminatory training procedure (i.e. it directly learns the class boundaries and does not *explicitly* build a separate model for each class), the implicit model corresponds to a GMB model, where we assume that for each class,

- $p(x \mid \Omega^I)$ is a mixture of spherically symmetric Gaussians ($\Sigma_j^I = \sigma^2 I$), where the largest term dominates the mixture for any given x, and
- α_j^I are equal and Q^I is proportional to $p(\Omega^I)$.

This is also the implicit model underlying any classifier which makes its classification decision based on the Euclidean distance measure between a feature vector and a set of prototype vectors (e.g. a VQ or k-Means (see section 3.1) clustering followed by classification based on (6.5)). In fact, we can reduce a GMB classifier to any VQ clustering based classifier which uses a quadratic distance measure for discrimination by making appropriate assumptions about the structure of the covariance matrices.

6.2 Regularized GMB classifiers

In the previous section, we presented GMB classifiers and WTA approximations to a GMB classifier. GMB classifiers (and WTA approximations) suffer from a quadratic increase of the number of parameters with input dimensionality. A mixture model with Q Gaussians has $O(Q * n^2)$ parameters where n is the number of input variables. This exacerbates the "*curse of dimensionality*" (a phrase attributed to Bellman (1961)) for high dimensional data sets. Data is very sparse in high dimensional spaces and model parameter estimates from small data sets are likely to be inaccurate. We use *regularization* procedures to constrain and/or reduce the effective number of model parameters and improve the generalization performance of the model. In this section, we discuss different methods of regularizing Gaussian mixture classifiers.

One of the standard techniques used to regularize Gaussian mixture models is to

assume that all the covariance matrices are diagonal or spherically symmetric (Nowlan 1991, Ahmad & Tresp 1993, Stensmo & Sejnowski 1995). The diagonal assumption has been effectively used for classification tasks (Nowlan 1991) and for solving the missing data problem (Ahmad & Tresp 1993). The spherically symmetric assumption has been widely used for classification and regression tasks (Stensmo & Sejnowski 1995, Nowlan 1991). These assumptions greatly reduce the number of parameters of the mixture models. Stensmo and Sejnowski (1995) and Nowlan (1991) have argued that the loss in flexibility due to the diagonal or spherical symmetry assumptions can be compensated for by using more mixture components. However, for many tasks, the diagonal or spherically symmetric assumptions are not valid because of correlations among the data variables.

In this section, we first describe a scheme for regularizing Gaussian mixture densities which also provides numerical stability to the EM algorithm. We then present another technique which prunes eigen-directions from the mixture discriminant functions and can be used in tandem with the former technique for regularization.

6.2.1 Bounding the covariance matrices

A problem with Gaussian mixture models is that, the likelihood function has a maximum whenever any of the covariance matrices is singular. One way to obtain numerical stability (avoid singularities) is to impose an artificial lower bound on the volume elements (determinants of covariance matrices) of each Gaussian (Ghahramani 1994, Ormoneit & Tresp 1995) during parameter estimation. This is achieved by adding a small diagonal matrix $\epsilon I_{n\times n}$ to each covariance matrix in each iteration of the EM algorithm. Adding $\epsilon I_{n\times n}$ avoids singularities and also regularizes the mixture model.

Ormoneit and Tresp (1995) define a Bayesian prior distribution $p(\theta)$ for the parameters θ of a mixture of Gaussians model and build Bayes classifiers. They derive EM update rules for maximizing the *a posterior* parameter probability. If $\mathcal{X}$ denotes a sample data set, the *a posterior* parameter probability is

$$p(\theta \mid \mathcal{X}) = \frac{p(\theta)\, p(\mathcal{X} \mid \theta)}{p(\mathcal{X})} \quad .$$

They implement the prior for the covariance matrices and obtain a modified EM update rule which corresponds to adding a diagonal matrix $\epsilon I_{n\times n}$ in each iteration. They note that ϵ introduces a bias which favors large variances. They vary ϵ over a range of values and choose the value for which the validation set classification error is the least.

In our experiments, we add an $\epsilon I_{n\times n}$ matrix to each covariance matrix in the M-step of every EM iteration. We also add ϵ to (and renormalize) all the posterior probabilities of Gaussian component membership during each iteration of EM to further regularize the model. Adding ϵ to posteriors prevents the Gaussians from being too far from all data points[1]. We choose the value of ϵ by varying it over a range of values and picking the value for which the validation set classification error is the least.

In the next subsection, we discuss our method for regularizing GMB classifiers. We still use the ϵ approach described above to enforce stability of the EM algorithm. We also constrain the discriminant functions to only look at certain directions of each covariance matrix to improve generalization performance.

6.2.2 Pruning eigen-directions from mixture discriminant functions

In this subsection, we will present our method of regularizing GMB classifiers. For each eigen-direction of each covariance matrix in the mixture discriminant functions (6.3), we empirically estimate the classification error induced by pruning that direction. For each covariance matrix in each discriminant function, we prune those p eigen-directions which induce the least classification error when removed. Thus, we remove p eigen-directions from each Gaussian such that the pruning induces the least bias (based on empirical estimates of the bias). Removing eigen-directions reduces the overall model variance and can improve the generalization performance. This pruning procedure is similar to the principal components pruning (PC pruning) (Levin *et al.* 1994) used for regularizing regression models.

In section 6.1, we assumed that the class conditional densities of the feature vectors

[1] I am grateful to Zoubin Ghahramani (MIT) for pointing this out.

x are mixtures of Gaussians and obtained the discriminant functions

$$\begin{aligned}\hat{\delta}^I(x) &= p(\Omega^I) \sum_{j=1}^{Q^I} \frac{\alpha_j}{(2\pi)^{n/2}\sqrt{|\Sigma_j^I|}} \exp\left[-\frac{1}{2}(x-\mu_j^I)^T \Sigma_j^{I\,-1}(x-\mu_j^I)\right] \\ &= p(\Omega^I) \sum_{j=1}^{Q^I} \frac{\alpha_j}{(2\pi)^{n/2}\sqrt{\prod_{i=1}^n \lambda_{ji}^I}} \exp\left[-\frac{1}{2}(x-\mu_j^I)^T \left(\sum_{i=1}^n \frac{e_{ji}^I e_{ji}^{I\,T}}{\lambda_{ji}^I}\right)(x-\mu_j^I)\right]\end{aligned} \tag{6.6}$$

where μ_j^I and Σ_j^I are the means and covariance matrices for the j^{th} component Gaussian. e_{ji}^I and λ_{ji}^I are the orthonormal eigenvectors and eigenvalues of Σ_j^I (ordered such that $\lambda_{j1}^I \geq \ldots \geq \lambda_{jn}^I$). In (6.6), we have written the Mahalanobis distance in terms of the eigenvectors.

We assume that we do not have enough data points to accurately estimate the discriminant functions (6.6). We are interested in constraining or regularizing the model to improve generalization performance. We ask the question; "which directions of a covariance matrix are most relevant for classification?". We measure the relevance (*saliency*) of an eigen-direction e_{ji}^I by removing the corresponding term in the mixture discriminant function (6.6) and computing the classification error for a validation data set with the modified discriminant functions. For each mixture component for each class, we prune p eigen-directions with the least saliency.

The **GMB-pruned** classification algorithm is as follows:

- For each class Ω^I, $i = 1, \ldots, K$, we train a mixture model of Q^I Gaussians using the EM algorithm. Thus, we obtain estimates of the discriminant functions (6.6).
- For each component j, $j = 1, \ldots, Q^I$, of the mixture discriminant function (6.6) for class Ω^I, $I = 1, \ldots, K$,
  - for each eigen-direction $l = 1, \ldots, n$,

* prune the terms corresponding to the l^{th} eigen-direction from the Mahalanobis distance and the determinant in (6.6),

$$\frac{\alpha_j}{(2\pi)^{n/2}\sqrt{\prod_{i\neq l}^{n}\lambda_{ji}^{I}}}\exp\left[-\frac{1}{2}(x-\mu_j^I)^T\left(\sum_{i\neq l}^{n}\frac{e_{ji}^I e_{ji}^{I\,T}}{\lambda_{ji}^I}\right)(x-\mu_j^I)\right].$$

Modify the discriminant function $\hat{\delta}^I(x)$, for class Ω^I (6.6) replacing the the j^{th} component Gaussian with the above expression,

$$p(\Omega^I)\sum_{k\neq j}^{Q^I}\frac{\alpha_k}{(2\pi)^{n/2}\sqrt{\prod_{i=1}^{n}\lambda_{ki}^I}}\exp\left[-\tfrac{1}{2}(x-\mu_k^I)^T\left(\sum_{i=1}^{n}\frac{e_{ki}^I e_{ki}^{I\,T}}{\lambda_{ki}^I}\right)(x-\mu_k^I)\right]+$$
$$p(\Omega^I)\frac{\alpha_j}{(2\pi)^{n/2}\sqrt{\prod_{i\neq l}^{n}\lambda_{ji}^I}}\exp\left[-\tfrac{1}{2}(x-\mu_j^I)^T\left(\sum_{i\neq l}^{n}\frac{e_{ji}^I e_{ji}^{I\,T}}{\lambda_{ji}^I}\right)(x-\mu_j^I)\right] \quad (6.7)$$

Compute the validation set classification error using the discriminant functions (6.6) for all classes except Ω^I and the discriminant function (6.7) for class Ω^I. Define the *saliency* s_l of the l^{th} eigen-direction as the classification error obtained for the validation set.

* Let E_j^I denote the set of p directions with the least saliency and S_j^I denote the set of $n-p$ directions with the largest saliency.

- For each class Ω^I, for each component j, $j=1,\ldots,Q^I$ of the discriminant function $\hat{\delta}^I(x)$, prune the p directions with the least saliency (elements of E_j^I). The modified discriminant functions are

$$\hat{\rho}^I(x)=p(\Omega^I)\sum_{j=1}^{Q^I}\frac{\alpha_j}{(2\pi)^{m/2}\sqrt{\prod_{i\in S_j^I}\lambda_{ji}^I}}\exp\left[-\frac{1}{2}(x-\mu_j^I)^T\left(\sum_{i\in S_j^I}\frac{e_{ji}^I e_{ji}^{I\,T}}{\lambda_{ji}^I}\right)(x-\mu_j^I)\right]. \quad (6.8)$$

- Classify inputs using the above discriminant functions.

We choose the value of p by cross-validation over the validation set. To reduce the complexity of cross validation search, we use the same value of p (number of directions pruned) for all mixture component for all classes. For each class and for each mixture component, the pruning procedure can be improved by the following technique.

- Prune the direction e_l with the least saliency.
- Re-estimate the saliencies of the other directions e_k $(k \neq l)$ as the classification error induced by pruning e_k in addition to e_l.
- Iterate the above steps p times, pruning the least saliency direction and re-estimating saliencies with the modified discriminant functions (*which do not have the newly pruned direction*).

This procedure can generate better estimates of the classification error induced by pruning p eigen-directions than the GMB-pruned algorithm, since it computes the saliencies by pruning *combinations* of directions. However, this procedure is $O(K * Q^I * n^2)$ which is prohibitively expensive[2].

6.3 Experimental Results

In this section we compare the different mixture models discussed above and a multi layer perceptron (MLP) for two speech phoneme classification tasks. The comparison measure is the classification accuracy. We divided the data sets into separate training, validation and test sets. We varied the free parameters of different algorithms over a range of values, and chose the value for which the validation set error was the least.

We trained several three layered MLPs (feedforward neural networks (see section 2.2 or (Hertz *et al.* 1991))) with different number of nodes in the hidden layer. The MLPs had sigmoid non-linearities in the hidden layer with asymptotes 0 and 1. Each network had K (number of classes) binary target outputs indicating class membership. The networks were trained using a conjugate gradient descent (see section 2.2.3 and (Press *et al.* 1987)) optimization scheme to minimize the mean squared error between the targets and the network outputs. We monitored the validation set error while training the MLPs

[2]For our data sets described in the next section, we estimated a training time of several weeks to use this procedure to prune eigen-directions.

and stopped training when the validation set error started to increase (*early stopping*; (Hertz *et al.* 1991)).

We trained several GMB classifiers using the EM algorithm, varying the number of components, the number of directions pruned (section 6.2.2) and the regularization parameter ϵ (section 6.2.1). We also trained GMB classifiers with a diagonal covariance approximation and the LVQ model. For all algorithms, we selected the value of free parameters by varying them over a range of values and picking the value for which the validation set error was the least.

6.3.1 TIMIT data

The first task is the classification of 12 monothongal vowels from the TIMIT database (Fisher & Doddington 1986). Each feature vector consists of the lowest 32 DFT coefficients, time-averaged over the central third of the vowel. This is the same data set used in section 5.1.2 and section 5.3.1 earlier in this document. We partitioned the data three times into different training sets (1200 vectors), validation sets (408 vectors), and test sets (408 vectors). The training sets contained 100 examples of each class. We report the *averaged* (over the three different partitions) test set classification accuracies with 2σ error bars where σ^2 is the variance across the three test sets.

We varied the values of free parameters over a range of values and chose that value which generated the least average (over the three different partitions) validation set classification error. We varied the number of hidden layer nodes of the MLP from 10 to 50 in increments of 5. For GMB classifiers (GMB-full and GMB-diagonal), we varied the number of mixture components, Q, from 1 to 5 and ϵ from 0.1 to 10^{-7}. Figure 6.2 shows a plot of the average validation set classification accuracy for the GMB-full model, for different values of ϵ for three component mixtures ($Q = 3$). For GMB-full, we obtained the least (average) validation set error with $Q = 3$ and $\epsilon = 0.01$. For a GMB-pruned model with this configuration, ($Q = 3$, $\epsilon = 0.01$), we varied the number of directions pruned from 2 to 27 in increments of 5 (in other words, we conducted experiments *retaining* 5 to 30 dimensions in increments of 5). For the LVQ, we varied

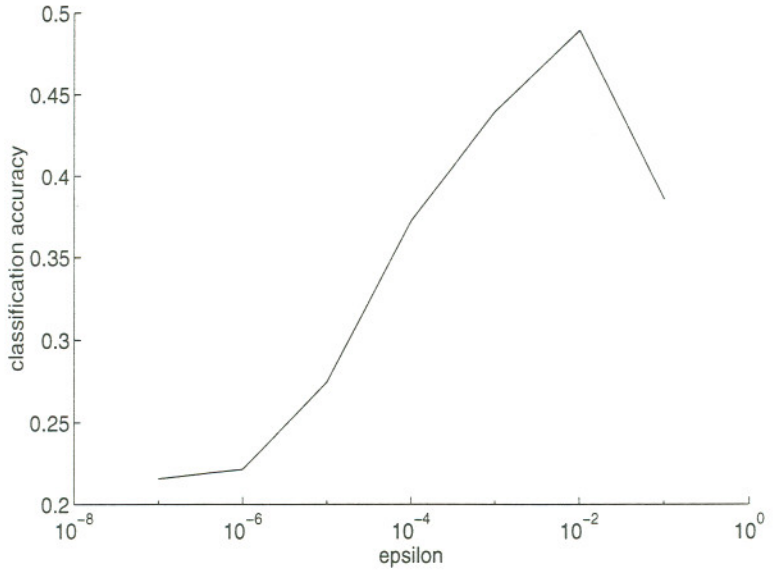

Figure 6.2: This plot shows the variation of the average classification accuracies of GMB classifiers with 3 component mixtures and full covariance matrices with respect to ϵ. Each point is the average value of the classification accuracies for the three validation sets.

the number of components from 15 to 90 in increments of 5.

Table 6.1 shows the results obtained with different algorithms. Since this data set contains very few data points per class, we expect regularization algorithms to perform well. For comparison, a GMB classifier (GMB-full) with 3 component mixtures and with a very small ϵ ($\epsilon = 0.000001$) obtained a classification accuracy of only 21.2%. We were unable to train a GMB classifier with zero ϵ (we obtained singular covariance matrices). We note from Table 6.1 that the difference in classification accuracies among the MLP and GMB algorithms is not statistically significant. A GMB with diagonal approximation has nearly four times the model variance as the other algorithms, and LVQ performs the worst. As noted above, we were unable to train totally unregularized mixture models with 3 components for this data. The rank reduction algorithm (GMB-pruned) prunes around 6% of the eigen-directions of each Gaussian and obtains a comparable accuracy to the full rank model.

Table 6.1: The averaged **test set** classification accuracies for the TIMIT vowels data for different algorithms with 2σ error bars. For GMB-pruned algorithm, "30-D" indicates that 30 of the original 32 eigen-directions were retained for each Gaussian.

ALGORITHM	ACCURACY
MLP (40 nodes in hidden layer)	45.8 ± 3.4%
GMB-full (3 components; $\epsilon = 0.01$)	46.7 ± 3.0%
GMB-pruned (3 components; 30-D; $\epsilon = 0.01$)	43.5 ± 3.3%
GMB-diagonal (2 components; $\epsilon = 0.00001$)	44.3 ± 7.5%
LVQ (70 cells)	38.5 ± 4.8%

6.3.2 CENSUS data

The next task we experimented with was the classification of 9 vowels (found in the utterances of the days of the week). The data was drawn from the CENSUS speech corpus (Cole, Novick, Burnett, Hansen, Sutton & Fanty 1994). Each feature vector was 70 dimensional (perceptual linear prediction (PLP) coefficients (Hermansky 1987) over the vowel and surrounding context). We partitioned the data into a training set (8997 vectors), a validation set (1362 vectors) for model selection, and a test set (1638 vectors). The training set had close to a 1000 vectors per class.

We varied the values of free parameters of algorithms over a range of values and chose those values which obtained the least validation set classification error. We varied the number of hidden layer nodes of MLPs from 10 to 90 in increments of 5. We varied the number of mixture components of GMB classifiers (with full and diagonal covariance matrices) from 1 to 6 and ϵ from 0.01 to 0.0001. For GMB-full, we obtained the largest classification accuracy for the validation set with 4 mixture components and $\epsilon = 0.01$. For the GMB-pruned algorithm, we used 4 mixture components and $\epsilon = 0.01$, and varied the number of directions pruned for each Gaussian from 5 to 65. For LVQ, we varied the number of components from 10 to 90 in increments of 5.

Table 6.2 gives a summary of the classification accuracies obtained using the different algorithms. This data set has a lot more data points per class than the TIMIT data set. The best accuracy is obtained by an MLP. We were unable to train an unregularized

Table 6.2: The **test set** classification accuracies for the CENSUS data for different algorithms.

ALGORITHM	ACCURACY
MLP (80 nodes in hidden layer)	88.2%
GMB-full (4 components; $\epsilon = 0.01$)	86.0%
GMB-pruned (4 components; 65-D; $\epsilon = 0.01$)	81.8%
GMB-diagonal (6 components; $\epsilon = 0.01$)	79.6%
LVQ (55 cells)	67.3%

mixture model (GMB-full with $\epsilon = 0$) with 4 components (some of the covariance matrices became singular after a few EM iterations). We note that the accuracies obtained by the full rank mixture model (GMB-full) and the MLP are very close. The reduced rank model (GMB-pruned) obtains a lower accuracy than GMB-full or the MLP. Both the regularized GMB models (using ϵ and reducing the rank) obtain higher accuracies than GMB-diagonal and LVQ. We hypothesize that the diagonal or spherically symmetric assumption for the covariance matrices was not appropriate for this data set. The rank reduction procedure prunes roughly 7% (chosen by cross validation) of the eigen-directions of each covariance matrix and does not obtain a high classification accuracy. We hypothesize that estimating the saliencies of eigen-directions by pruning combinations of directions as sketched in section 6.2.2 can generate better results, since the estimates of the error induced by pruning will be more accurate. We were unable to perform experiments with pruning combinations of directions, since it entailed prohibitive computation time requirements (on the order of several weeks for one experiment).

6.4 Discussion

In this chapter, we presented Gaussian mixture Bayes (GMB) classifiers, where we assume that the class conditional densities are mixtures of Gaussians and derive Bayes discriminant functions. We showed that winner-take-all approximations to GMB classifiers are related to hard clustering based classifiers like the learning vector quantization

(LVQ) algorithm.

We discussed several methods of regularizing GMB classifiers:

- The models are regularized by assuming that all covariance matrices are diagonal (GMB-diagonal) or spherically symmetric (as in LVQ).

- Following (Ghahramani & Jordan 1994*a*), a constant diagonal matrix ($\epsilon I_{n\times n}$) is added to each covariance matrix in each iteration of the EM algorithm (for training the mixture models). This enforces numerical stability and regularizes the mixture model. Applying a Bayesian prior to the parameters, Ormoneit *et al* (1995) obtain the same EM update rule. In our implementation, we also add ϵ to (and renormalize) the posterior probabilities of Gaussian component membership during each iteration of the EM algorithm as in (Ghahramani & Jordan 1994*a*).

- We propose a new method of regularizing GMB classifiers. Using ideas from principal components pruning (PC pruning; (Levin *et al.* 1994)), we regularize mixture discriminant functions by pruning eigen-directions of Gaussian components which induce the least classification error (empirically measured for a validation set) when pruned. This can improve the generalization performance by decreasing the overall model variance (at the expense of added bias).

We presented experimental results for two speech phoneme classification tasks. Both the regularization schemes obtain higher classification accuracies than an unregularized GMB model, a diagonal approximation (GMB-diagonal) or a spherically symmetric approximation (LVQ). We conclude that

- the diagonal or spherically symmetric covariance matrices assumption was invalid for the data sets we experimented with, since GMB classifiers with full covariance matrices performed better than GMB with diagonal or spherically symmetric assumptions. This suggests that the data variables are locally (in the input space) correlated.

- Some regularization is necessary for training the mixture models to prevent singular covariance matrices and to prevent over-parametrization in high dimensional spaces. Regularizing the mixture models by bounding the covariance matrices using $\epsilon I_{n\times n}$ while training seems to be the best technique. Pruning eigen-directions of covariance matrices does not perform as well. The pruning technique may be improved by pruning *combinations* of eigen-directions while generating saliency estimates.

In the next chapter, we describe local linear regression algorithms based on a mixture of Gaussians model for the joint density of input and output variables.

Chapter 7

Gaussian mixture models for regression

The goal of regression analysis is to predict values of response variables y, given observed values of predictor variables x. In this chapter, we describe Gaussian mixture models for regression. We model the joint density of the predictor and response variables as a mixture of Gaussians, and derive the least squares regression estimate $E[y \mid x]$. This is also the approach taken by Ghahramani and Jordan (1994*a*, 1994, 1994*b*). The regression $E[y \mid x]$ is a local linear function of the predictor variables; a weighted sum of linear models, where the weights are the probabilities of membership to the component Gaussians of the mixture model.

We propose two new ways of regularizing the local linear regression function:

- local ridge regression, where we shift the local prediction matrix, inducing bias and possibly reducing model variance, and
- principal components (PC) pruning (similar to (Levin *et al.* 1994)), where we prune those directions of each prediction matrix which induce the least additional error when pruned.

We present results of experiments predicting the housing prices in Boston, the average monthly sunspots count and a chaotic time series. The best technique was task dependent. However, regularizing the local linear predictors seemed to be useful whenever the problem was high dimensional and the sample size was small.

In the next section, we will define regression analysis and describe the standard linear model. We will then present Gaussian mixture models for regression and techniques

for regularizing them. Finally we present empirical results comparing all the models discussed here.

7.1 Introduction

Regression analysis (Draper & Smith 1981, Breiman *et al.* 1984, Seber & Wild 1989) predicts the values of m random variables called *response variables* (denoted here by a vector $y \in \mathcal{R}^m$) given the values of n other random variables called *regressor* or *predictor variables* (denoted here by a vector $x \in \mathcal{R}^n$). The standard model (Draper & Smith 1981, Seber & Wild 1989) is

$$y = f(x) + \epsilon, \tag{7.1}$$

where $f(\cdot)$ is a mapping from $\mathcal{R}^n$ to $\mathcal{R}^m$, and ϵ is generated by an unobservable zero-mean noise process (i.e $E[\epsilon] = 0$). The components of x are observable or functions of observable variables.

The response vector y is predicted using a function $g(x)$ parametrized by θ. The accuracy of prediction is usually measured by the mean squared error

$$\mathcal{E} = E\left[\|y - g(x)\|^2\right] , \tag{7.2}$$

where the expectation $E[\cdot]$ is over both x and ϵ. The mean squared error (7.2) is minimized by the function $g^*(x) = E[y \,|\, x] = f(x)$, the conditional expectation of the response, which is called the regression of y on x (Draper & Smith 1981, Breiman *et al.* 1984). Given N sample pairs of the predictor and response variables $\{x_i, y_i \,|\, i = 1, \ldots, N\}$, the *least squares estimate* of θ minimizes the average squared error $S(\theta)$ which is an estimate of $\mathcal{E}$,

$$\begin{aligned} \hat{\theta} &= \operatorname*{argmin}_{\theta} S(\theta) \\ &\equiv \operatorname*{argmin}_{\theta} \frac{1}{N} \sum_{i=1}^{N} \|y_i - g(x_i;\theta)\|^2 , \end{aligned} \tag{7.3}$$

If we assume that the noise vectors ϵ_i (corresponding to x_i, y_i pairs) are independent and identically Gaussian distributed with a zero mean and a covariance matrix $\sigma^2 I_{m\times m}$,

then the least squares estimate of θ is also the maximum likelihood estimate (Draper & Smith 1981, Seber & Wild 1989). Let $\mathcal{Y} = \{y_1, \ldots, y_N\}$ denote the set of sample values of y. The likelihood of $\mathcal{Y}$ given our model is

$$\begin{aligned} p(\mathcal{Y} \mid \theta, \sigma^2) &= \prod_{i=1}^{N} \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left[-\frac{\|y_i - g(x_i;\theta)\|^2}{2\sigma^2}\right] \\ &= \frac{1}{(2\pi\sigma^2)^{Nm/2}} \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{N}\|y_i - g(x_i;\theta)\|^2\right] \\ &= \frac{1}{(2\pi\sigma^2)^{Nm/2}} \exp\left[-\frac{1}{2\sigma^2} S(\theta)\right] \quad . \end{aligned} \tag{7.4}$$

The parameter estimates which maximize $p(\mathcal{Y} \mid \theta, \sigma^2)$ are $\theta = \hat{\theta}$ (the least squares estimate) and $\hat{\sigma^2} = S(\hat{\theta})/m$ (Seber & Wild 1989).

7.1.1 Linear regression model

In the standard linear model (Draper & Smith 1981, Breiman *et al.* 1984, Seber & Wild 1989), $g(x)$ is restricted to $g(x) = \omega x$ where $\omega \in \mathcal{R}^{m \times n}$. The optimal least squares estimate of ω is the solution of the Widrow-Hoff equations (Widrow & M. Hoff, Jr. 1960, Haykin 1994)

$$\omega R = r \quad , \tag{7.5}$$

where $R \equiv E[xx^T]$ is the auto-correlation matrix of the predictor vectors x and $r \equiv E[yx^T]$ denotes the cross-correlation matrix between the response and predictor variables. Given N sample pairs of predictor and response variables $\{x_i, y_i \mid i = 1, \ldots, N\}$, the optimal (least squared error) ω is

$$\begin{aligned} \hat{\omega} &= \hat{r}\hat{R}^{-1} \\ &\equiv \left[\frac{1}{N}\sum_{i=1}^{N} y_i x_i^T\right]\left[\frac{1}{N}\sum_{i=1}^{N} x_i x_i^T\right]^{-1} \quad , \end{aligned} \tag{7.6}$$

where $\hat{r}$ is the sample cross correlation matrix between the predictor and response variables and $\hat{R}$ is the sample predictor auto-correlation matrix.

When the predictor variables are statistically dependent on each other, the auto-correlation matrix $\hat{R}$ can be close to singular, resulting in unstable parameter estimates.

Ridge regression (Draper & Smith 1981), a method originally conceived to tackle this problem, changes the least squares solution $\hat{\omega}$ by shifting the predictor correlation matrix from $\hat{R}$ to $\hat{R} + \rho I_{n\times n}$,

$$\hat{\omega} = \hat{r}\left[\hat{R} + \rho I_{n\times n}\right]^{-1} , \tag{7.7}$$

where ρ is a parameter. As ρ is increased, the estimates become smaller in absolute value. In general, this biases the estimate, and decreases the model variance. For the optimal value of ρ, this effect can decrease the mean squared (generalization) error, even when the auto-correlation matrix is not close to singular. In practice, the optimal value of ρ is estimated by varying it over a range of values and picking the value for which the averaged square error $S(\theta)$ for a separate *validation data set* is the least.

For linear regression, the least squares estimate, $E[y\,|\,x]$ is derived by assuming a functional form for $g(x)$. Another approach is to assume a parametric model of the joint density of the predictors and the response variables and directly compute the optimal least squares solution $E[y\,|\,x]$. In the next section, we describe a local linear regression model based on a mixture of Gaussians model for the joint density.

7.2 The Gaussian mixture regression (GMR) model

We model the joint density of the the response variables $y \in \mathcal{R}^m$ and the predictors $x \in \mathcal{R}^n$ as a mixture of Q multivariate $(m+n)$-dimensional Gaussian functions,

$$p(z) = \sum_{j=1}^{Q} \frac{\alpha_j}{(2\pi)^{(m+n)/2}\sqrt{|\Sigma_j|}} \exp\left[-\frac{1}{2}(z-\mu_j)^T \Sigma_j{}^{-1} (z-\mu_j)\right] , \tag{7.8}$$

where $z = (y, x)^T$ and μ_j and Σ_j denote the means and covariance matrices of the Gaussian components. The conditional density $p(y\,|\,x)$ is also a mixture of Gaussians and the regression $E[y\,|\,x]$ is

$$\begin{aligned} E[y\,|\,x] &= \int dy\; y\; p(y\,|\,x) \\ &= \frac{\int dy\; y\; p(y, x)}{p(x)} \end{aligned}$$

$$
\begin{aligned}
&= \frac{\int dy\; y\, p(y,x)}{\int dy\; p(y,x)} \\
&= \sum_{j=1}^{Q} h_j(x) \left[\mu_j^y + \Sigma_j^{yx}\Sigma_j^{xx-1}\left(x-\mu_j^x\right)\right] \quad , \qquad (7.9)
\end{aligned}
$$

where the weighting function $h_j(x)$,

$$
h_j(x) \equiv \frac{\frac{\alpha_j}{(2\pi)^{n/2}\sqrt{|\Sigma_j^{xx}|}} \exp\left[-\frac{1}{2}(x-\mu_j^x)^T\,\Sigma_j^{xx-1}\,(x-\mu_j^x)\right]}{\sum_{k=1}^{Q} \frac{\alpha_k}{(2\pi)^{n/2}\sqrt{|\Sigma_k^{xx}|}} \exp\left[-\frac{1}{2}(x-\mu_k^x)^T\,\Sigma_k^{xx-1}\,(x-\mu_k^x)\right]} \quad , \qquad (7.10)
$$

denotes the probability that the j^{th} Gaussian component of the marginal predictor density $p(x)$ generated the predictor vector x. Here we use y and x superscripts to denote subvectors and submatrices which correspond to the response and predictor variables. For example, Σ_j is divided into

$$
\Sigma_j = \begin{pmatrix} \Sigma_j^{yy} & \Sigma_j^{yx} \\ \Sigma_j^{xy} & \Sigma_j^{xx} \end{pmatrix}
$$

where

- Σ_j^{yy} denotes the $m \times m$ submatrix formed by the first m rows and columns of Σ_j,
- Σ_j^{yx} denotes the $m \times n$ submatrix formed by the first m rows and the last n columns of Σ_j,
- Σ_j^{xy} denotes the $n \times m$ submatrix formed by the last n rows and the first m columns of Σ_j, and
- Σ_j^{xx} denotes the $n \times n$ submatrix formed by the last n rows and columns of Σ_j.

Similarly, μ_j^y and μ_j^x denote the subvectors of the means corresponding to the response and predictor variables respectively.

Ghahramani and Jordan (1994*b*, 1994*a*, 1994) use a mixture of Gaussians model for the joint density of predictor and response variables to derive the regression function (7.9). Tresp *et al* (1994) mention using a mixture model to derive the regression function. Marroquin (1995) uses a soft local linear model for regression which is similar to (7.9).

However, Marroquin assumes that the covariance matrices of the mixture model are spherically symmetric. The regression function (7.9) (also see (Ghahramani & Jordan 1994*b*, Ghahramani & Jordan 1994*a*)) is a weighted sum of standard linear regression functions discussed in section 7.1 as follows:

$$\begin{aligned} E[y \mid x] &= \sum_{j=1}^{Q} h_j(x) \left[\mu_j^y + \Sigma_j^{yx} \Sigma_j^{xx\,-1} \left(x - \mu_j^x \right) \right] \\ &= \sum_{j=1}^{Q} h_j(x) \left[\omega_j \, \tilde{x} \right] \quad , \end{aligned} \tag{7.11}$$

where $\tilde{x} \equiv (x, 1)$ is an $(n+1)$ dimensional column vector and $\omega_j \in \mathcal{R}^{m \times (n+1)}$ is the j^{th} local prediction matrix

$$\omega_j \equiv \left[\Sigma_j^{yx} \Sigma_j^{xx\,-1}, \mu_j^y - \Sigma_j^{yx} \Sigma_j^{xx\,-1} \mu_j^x \right] \quad . \tag{7.12}$$

By substitution, we verify that ω_j is the solution to the Widrow-Hoff equations

$$\omega_j \, R_j = r_j \quad , \tag{7.13}$$

where

$$\begin{aligned} R_j &\equiv E[h_j(x) \, \tilde{x}\tilde{x}^T] \\ &= \begin{bmatrix} E[h_j(x) \, xx^T] & E[h_j(x) \, x] \\ E[h_j(x) \, x^T] & 1 \end{bmatrix} \\ &= \begin{bmatrix} \Sigma_j^{xx} + \mu_j^x \mu_j^{x\,T} & \mu_j^x \\ \mu_j^{x\,T} & 1 \end{bmatrix} \end{aligned} \tag{7.14}$$

and

$$\begin{aligned} r_j &\equiv E[h_j(x) \, y\tilde{x}^T] \\ &= \left(E[h_j(x) \, yx^T], \mu_j^y \right) \\ &= \left(\Sigma_j^{yx} + \mu_j^y \mu_j^x, \mu_j^y \right) \end{aligned} \tag{7.15}$$

are the weighted[1] autocorrelation and cross-correlation matrices of $\tilde{x}$ for the j^{th} Gaussian component. Recall that the weighting $h_j(x)$ denotes the probability that the j^{th} Gaussian component of the marginal predictor density $p(x)$ generated the predictor vector x. Ghahramani and Jordan (1994*b*, 1994*a*) note that "the mixture of Gaussians competitively partitions the input space, and learns a linear regression surface in each partition". They also note the similarity of the model presented above to *classification and regression trees* (Breiman *et al.* 1984) and the mixture of experts (Jacobs *et al.* 1991, Jordan & Jacobs 1994) which also soft partition the data to build local models.

The *Gaussian mixture regression* (**GMR**) algorithm is:

- Train a mixture of Gaussians model for the joint density of $z = (y, x)$ using the EM algorithm (see appendix C).
- After training, for any given x, compute the prediction of y using (7.9).

The GMR algorithm can generate unstable parameter estimates whenever *any* of the local auto-covariance matrices Σ_j is close to singular. As discussed in chapter 6 (see section 6.2), the log likelihood function for a mixture of Gaussians has a maximum whenever any of the covariance matrices is singular. Moreover, the number of parameters, ($O[Q * (n+m)^2]$ parameters), scales faster than linearly with input dimensionality. This can lead to inaccurate parameter estimates for small data sets of high dimensional data.

One way to ensure numerical stability of parameters is to add a constant diagonal $\epsilon I_{(m+n)\times(m+n)}$ matrix in each EM iteration (see section 6.2). This is also a regularization scheme for mixture models. For all the results presented in this chapter for mixture models, we use this method. However, this does not solve the potential over-parametrization problem.

[1]The weightings are a result of the mixture density; e.g. Σ_j is a weighted auto-covariance matrix of *all* the predictor vectors, where the weighting for x, $h_j(x)$, is the posterior probability that the j^{th} component Gaussian generated x.

One way to reduce the number of parameters is to assume that the covariance matrices Σ_j are diagonal. The least squares regression (7.11) becomes

$$E[y \mid x] = \sum_{j=1}^{Q} h_j(x) \mu_j^y \quad , \tag{7.16}$$

a locally constant model of the data. We will refer to regression using (7.16) as the **GMR-const** algorithm. The regression function is a weighted average of the means of the response variables, where the weights represent the closeness to the corresponding predictor means. As Ghahramani (1994*a*) notes, this function has an identical form to normalized radial basis functions (Moody & Darken 1988, Poggio & Girosi 1990). The GMR-const procedure can generate high errors whenever the diagonal assumption is incorrect due to correlations between the predictor and response variables, which is often the case.

We propose two new ways of regularizing the local linear regression (7.11). The first method uses local ridge regression. The second method prunes the directions of each local linear predictor which contribute the least towards lowering the mean squared error.

7.2.1 GMR with local ridge regression

We apply *ridge regression* (see section 7.1) to each local prediction matrix in (7.11) (Leen 1995). Let $\hat{\omega}_j$, $\hat{R}_j$, and $\hat{r}_j$ denote estimates of ω_j, R_j and r_j based on a data set $\mathcal{Z} = \{(y_i, x_i), i = 1, \ldots, N\}$ after training a mixture model for the joint density of $z = (y, x)$. We change each local prediction matrix $\hat{\omega}_j$ by shifting the local predictor correlation matrix $\hat{R}_j$ from $\hat{R}_j$ to $\hat{R}_j + \rho I_{(n+1)\times(n+1)}$,

$$\hat{\omega}_j = \hat{r}_j \left[\hat{R}_j + \rho I_{(n+1)\times(n+1)} \right]^{-1} \quad , \tag{7.17}$$

where ρ is a parameter. Just as in linear ridge regression, increasing ρ makes the estimates smaller in magnitude, biases them, and decreases the model variance. For the optimal value of ρ, this can reduce the mean squared error. We estimate the optimal value of ρ by varying it over a range of values and choosing the value for which the averaged squared error $S(\theta)$ is the least for a validation data set. We call this procedure **GMR-ridge**.

7.2.2 GMR with principal components pruning

We now present another method of regularizing the GMR regression function (7.11), which computes the components of each local prediction matrix $\hat{\omega}_j$ along the eigen-directions of $\hat{R}_j$, and prunes those p components which induce the least bias when removed (Leen 1995). This procedure is an application of the principal components pruning (PC pruning) technique (Levin *et al.* 1994) applied to mixture regression models.

The GMR algorithm with PC pruning (**GMR-prune**) is as follows.

- Train a mixture of Gaussians model for the joint density of the response and predictor variables using the EM algorithm.
- Compute the local prediction matrices $\hat{\omega}_j$ (7.12) and the local auto-correlation matrices $\hat{R}_j$ (7.14).
- For each response variable y^k, $k = 1, \ldots, m$,
  - for each local predictor vector $\hat{\omega}_{jk}$, $j = 1, \ldots, Q$,
    - compute the components of $\hat{\omega}_{jk}$ along the eigen-directions of $\hat{R}_j$. Let $\{e_1, \ldots, e_{n+1}\}$ and $\{\lambda_1, \ldots, \lambda_{n+1}\}$ denote the orthonormal eigenvectors and corresponding eigenvalues of $\hat{R}_j$ respectively. Note the these eigenvectors span $\mathcal{R}^{n+1}$. Compute β_l, the component of $\hat{\omega}_{jk}$ along e_l,

$$\hat{\omega}_{jk} = \sum_{l=1}^{n+1} \beta_l e_l \equiv \sum_{l=1}^{n+1} (\hat{\omega}_{jk}^T e_l) e_l \quad , \tag{7.18}$$

      for $l = 1, \ldots, n+1$.

    - Define the *saliency*, s_l of the l^{th} component above as the local 1-component (for the response variable y^k) bias induced by setting $\beta_l = 0$,

$$\begin{aligned} s_l &= E_{h_j(x)} [\, \| \beta_l e_l^T \tilde{x} \|^2 \,] \\ &= E_{h_j(x)} [\, \beta_l^2 e_l^T \tilde{x} \tilde{x}^T e_l \,] \\ &= \beta_l^2 \, e_l^T E_{h_j(x)} [\, \tilde{x} \tilde{x}^T \,] e_l \\ &\approx \beta_l^2 \, e_l^T \hat{R}_j e_l \end{aligned}$$

$$= \beta_l^2 \lambda_l \ . \tag{7.19}$$

Here, $E_{h_j(x)}[\cdot]$ denotes an expectation with respect to the probability density function of membership of x to the j^{th} Gaussian component. Therefore, by definition, $\hat{R}_j$ is an estimate of $E_{h_j(x)}[\tilde{x}\tilde{x}^T]$. Since e_l is an eigenvector of $\hat{R}_j$, $e_l^T \hat{R}_j e_l = \lambda_l$. For each $l = 1, \ldots, n+1$, compute s_l.

* For the p directions with the least saliency, set $\beta_l = 0$. Compute the new (regularized) estimate of $\hat{\omega}_{jk}$ as

$$\hat{\omega}_{jk} = \sum_{l=1}^{n+1} \beta_l e_l \ . \tag{7.20}$$

This effectively prunes the p directions with the least saliency for the j^{th} local predictor for the k^{th} response variable.

- For each data point x, compute the prediction of y (7.11) using the regularized ω_j matrices.

We choose the number of directions to prune, p, by cross validation. We vary p over a range of values (including 0) and choose the value for which the validation data set error is the least.

7.3 Experimental results

In this section, we compare the local linear regression algorithms discussed above (GMR-const, GMR-ridge and GMR-prune) and a multi-layer perceptron (MLP or a feedforward neural network; see section 2.2) for three benchmark regression tasks: predicting the Boston housing prices, average monthly sunspot activity and a chaotic time series. Let $\{(y_i, x_i), i = 1, \ldots, N\}$ denote the *test set* for a task. For each algorithm, we compute the normalized error,

$$\mathcal{E}_{norm} \equiv \frac{\frac{1}{N}\sum_{i=1}^{N} \|y_i - g(x;\theta)\|^2}{\frac{1}{N}\sum_{i=1}^{N} \|y_i - \mu^y\|^2} = \frac{S(\theta)}{\mathrm{Var}[y]} \ , \tag{7.21}$$

the average squared error $S(\theta)$ normalized by the variance in the response variables. Here, $\mu^y = \frac{1}{N}\sum_{i=1}^{N} y_i$. A normalized error of 1.0 implies that the squared error is equal to that of a trivial predictor which always predicts the mean value of the response variables.

We varied the free parameters of all algorithms over a range of values and chose the value for which the validation set error is the least. For all the local linear models, we added a diagonal matrix $\epsilon I_{(m+n)\times(m+n)}$ to each local covariance matrix in each iteration of the EM algorithm. We trained multi-layer perceptrons (MLPs) with one hidden layer using a quasi-Newton BFGS (Press *et al.* 1987) optimization scheme.

7.3.1 Prediction of Boston housing prices

Our first task is the prediction of housing prices in Boston (the response variable) from 13 factors (the predictor variables) (Breiman *et al.* 1984, Moody & Yarvin 1992). We trained the algorithms with four different partitions of the data into training sets containing 380 vectors, validation sets containing 63 vectors and test sets containing 63 vectors. We varied the number of hidden layer nodes of the MLPs from 2 to 40 in increments of 2. For the local linear algorithms, we varied the number of mixture components from 1 to 10, and the regularization parameter ϵ, from 10^{-6} to 0.01. We varied the number of directions pruned using PC pruning from 0 to 12 and the ridge regression parameter ρ from 10^{-5} to 10. For all algorithms, we report results with those parameter values which obtained the least average (across the 4 permutations of training, validation and test sets) validation set error.

In Table 7.1, we report the average normalized test set errors (7.21) for different algorithms with 2σ error bars. Here σ^2 is the variance of the test set error across the four different permutations of training, validation and test sets.

Breiman *et al* report a normalized error $\mathcal{E}_{norm} \approx .22$ (in Table 8.8 on page 249 of (Breiman *et al.* 1984)) using the *classification and regression trees* (CART) algorithm. Moody and Yarvin (1992) train feedforward neural networks with different activation functions and report an error between $\mathcal{E}_{norm} = 0.15$ and $\mathcal{E}_{norm} = 0.20$. Our results are

Table 7.1: The average (across 4 different permutations of training, validation and test sets) normalized test set error (7.21) for the prediction of Boston housing prices using different algorithms. We chose the values of free parameters by cross validation over the average validation set error (see text for details). For the mixture models, we report the number of components, the regularization parameter ϵ, the number of directions pruned using principal components pruning p, and the ridge regression parameter ρ.

ALGORITHM	$\mathcal{E}_{norm}$
MLP (2 nodes in hidden layer)	$.20 \pm .19$
Linear Regression	$.28 \pm .13$
GMR (6 comps; $\epsilon = 10^{-6}$)	$.24 \pm .14$
GMR-pruned (6 comps; $\epsilon = 10^{-6}$; $p = 4$)	$.24 \pm .14$
GMR-ridge (6 comps; $\epsilon = 10^{-6}$; $\rho = 10^{-5}$)	$.24 \pm .14$
GMR-const (5 comps; $\epsilon = 10^{-6}$)	$.88 \pm .16$

comparable to all the results quoted above. We note from Table 7.1 that the difference between the error using linear regression and the error using an MLP or any of the GMR algorithms except GMR-const is statistically insignificant. Also, all the GMR algorithms (unregularized, using PC pruning and using ridge regression) perform comparably. The GMR-const procedure (which is related to radial basis functions; see section 7.2) performs the worst with a statistically significant margin.

7.3.2 Prediction of the average monthly sunspots count

Our next task is the prediction of the average monthly sunspot count in a given year from the values of the twelve previous years (Moody & Yarvin 1992). We use the data from the years 1712 to 1880 as the training set (169 data points), the data from the years 1881 to 1920 as the validation set (40 data points) and the data from the years 1921 to 1955 as the test set (35 data points).

We varied the number of hidden layer nodes of the MLPs from 2 to 7. We varied the number of mixture components of all the local linear algorithms from 1 to 10, and the regularization parameter ϵ, from 10^{-6} to 0.01. We varied the number of directions pruned using principal components pruning from 0 to 11 and the ridge regression parameter ρ

Table 7.2: The normalized prediction errors (7.21) for the test set for predicting the average monthly sunspot count using different algorithms.

ALGORITHM	$\mathcal{E}_{norm}$
MLP (3 nodes in hidden layer)	.116
Linear Regression	.118
GMR (6 comps; $\epsilon = 10^{-6}$)	.136
GMR-pruned (6 comps; $\epsilon = 10^{-6}$; 8-D)	.108
GMR-ridge (6 comps; $\epsilon = 10^{-6}$; $\rho = 0.01$)	.125
GMR-const (5 comps; $\epsilon = 10^{-6}$)	.411

from 10^{-5} to 10. For all algorithms, we report results with those parameter values which obtained the least validation set error. We summarize our results in Table 7.2.

For the same test set as ours, Moody and Yarvin (1992) obtain an error between $\mathcal{E}_{norm} = .105$ and $\mathcal{E}_{norm} = .111$ using feedforward neural networks (MLPs) with different activation functions. They note that the best test set error achieved by them or previous testers was about $\mathcal{E}_{norm} = .085$ (Moody & Yarvin 1992). Our results are comparable to theirs. The GMR-prune algorithm (which uses PC pruning) prunes 5 directions from each local prediction matrix, and seems to provide the best performance among all algorithms tested. Note that the unregularized local linear algorithm with 6 components (GMR) obtains a much larger error than linear regression. Both the regularization schemes (PC pruning and ridge regression) successfully reduce the error obtained by GMR. The GMR-const algorithm again performs worst obtaining nearly 4 times the error of any of the other algorithms.

7.3.3 Prediction of the Mackey-Glass time series

Our next task is to predict data points from a chaotic time series generated by integrating the Mackey-Glass differential-delay equation,

$$\frac{d}{dt}x(t) = a\frac{x(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \quad .$$

Table 7.3: Prediction errors (7.21) for the test set from the Mackey-Glass time series using different algorithms (see text for details).

ALGORITHM	$\mathcal{E}_{norm}$
MLP (38 nodes in hidden layer)	.02
Linear Regression	.31
GMR (60 comps; $\epsilon = 0.0001$)	.04
GMR-pruned (60 comps; $\epsilon = 0.0001$; 4-D)	.04
GMR-ridge (60 comps; $\epsilon = 0.0001$; $\rho = 10^{-6}$)	.04
GMR-const (60 comps; $\epsilon = 0.0001$)	.14

With $a = 0.2$, $b = 0.1$, and $\tau = 17$, the trajectory is chaotic and the time series is a benchmark for prediction algorithms (Farmer & Sidorowich 1987, Moody & Darken 1989, Moody & Darken 1988, Hartman & Keeler 1991). Following (Moody & Darken 1989, Moody & Darken 1988, Hartman & Keeler 1991), the goal is to predict $x(t+85)$ given $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$. We divided the training set containing 500 vectors used by Hartman and Keeler (1991) into a training set containing 350 vectors, and a validation set containing 150 vectors. We use the same test set as Hartman and Keeler (1991) which contains 500 vectors[2].

We varied the number of hidden layer nodes of the MLPs from 2 to 40 in increments of 2. We varied the number of mixture components of the local linear algorithms from 5 to 80 in increments of 5, and the regularization parameter ϵ, from 10^{-6} to 0.01. Figure 7.1 shows the variation of the normalized squared error for the validation set for GMR with 60 components and full covariance matrices, for different values of ϵ. We varied the number of directions pruned using PC pruning from 0 to 3. We obtained the least validation error with $Q = 60$ components, $\epsilon = 0.0001$ and with one direction pruned for every local prediction matrix. We used these values of Q and ϵ (60 and 0.0001 respectively) for all remaining experiments. We varied the ridge regression parameter ρ from 10^{-6} to 10. For all algorithms, we report results with those parameter values which obtained the least validation set error. We summarize our results in Table 7.3.

[2]I am grateful to Thorsteinn Rognvaldsson of OGI for providing me with this data set.

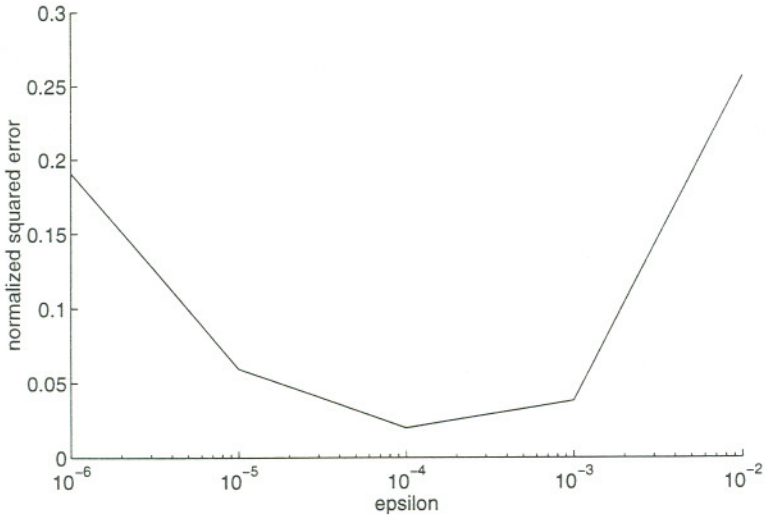

Figure 7.1: This plot shows the variation of the normalized squared error for the validation set for GMR with 60 components and full covariance matrices, with respect to ϵ.

Lapedes and Farber (1987) trained an MLP with 2 hidden layers (each containing 20 nodes with a sigmoidal activation function) and with 500 data points to do the above prediction. They obtained a *normalized prediction error* ($\sqrt{\mathcal{E}_{norm}}$) of 0.05. This corresponds to $\mathcal{E}_{norm} = 0.0025$. Moody and Darken (1989, 1988) used radial basis functions with about a 1000 Gaussian units and with 7 times more training data to obtain a comparable error. When they used around 100 Gaussian units, they obtained an error of 0.25 ($\mathcal{E}_{norm} = 0.0625$). Hartman and Keeler (1991) report an error of 0.08 ($\mathcal{E}_{norm} = 0.0064$) using RBFs with 300 units and sigmoid MLPs (2 hidden layers; 10 nodes in each).

We obtained a much higher error than the best results discussed above. However, we obtain a much lower error (with most of the algorithms) than RBFs with 100 Gaussian units (Moody and Darken (1989, 1988): $\mathcal{E}_{norm} \approx 0.06$ and Hartman and Keeler (1991): $\mathcal{E}_{norm} \approx 0.08$). The performance of our algorithms can probably be improved by using more parameters, since this data set does not have any noise.

Note from Table 7.3 that all the non-linear algorithms obtain a substantially lower

Table 7.4: The averaged **test set** classification accuracies for the TIMIT vowels data for different algorithms with 2σ error bars.

ALGORITHM	ACCURACY
MLP (40 nodes in hidden layer)	45.8 ± 3.4%
GMB-full (3 components; $\epsilon = 0.01$)	46.7 ± 3.0%
GMB-pruned (3 components; 30-D; $\epsilon = 0.01$)	47.3 ± 3.9%
GMB-diagonal (2 components; $\epsilon = 0.00001$)	44.3 ± 7.5%
LVQ (70 cells)	38.5 ± 4.8%
GMR-prune (1 component; $\epsilon = 10^{-4}$, 22-D)	43.2 ± 4.4%
GMR-ridge (1 component; $\epsilon = 10^{-4}$, $\rho = 10^{-5}$)	42.6 ± 3.9%
GMR-const (1 component; $\epsilon = 10^{-4}$)	8.3 ± 0.0%

error than linear regression, indicating a significant non-linearity in the problem. The MLP obtains the best performance. Both the regularization schemes (PC pruning and local ridge regression) for local linear regression obtain nearly the same error as the unregularized algorithm (GMR). This might be because the sample size was large enough to obtain good estimates of the 5 dimensional joint density. The GMR-const algorithm again obtained a lower error than any of the other local linear algorithms.

7.3.4 GMR applied to classification

In this subsection, we apply the GMR algorithms for *classification* tasks. We consider a one-of-K classification task as a regression task with K response variables. We denote the response vector of a predictor vector belonging to class c as $y_1 = 0, \ldots, y_{c-1} = 0, y_c = 1, y_{c+1} = 0, \ldots, y_K = 0$.

Our first task is the classification of 12 monothongal vowels from the TIMIT database (Fisher & Doddington 1986). There are 32 predictor variables belonging to 12 classes (12 response variables). This is the same data set used in sections 5.1.2, 5.3.1 and 6.3.1 of this document. We summarize our results using GMR algorithms in Table 7.4 along with our previous results with GMB classifiers. Please see chapter 6 for a description of GMB classification algorithms. We note that GMR-prune and GMR-ridge perform comparably to the GMB classifiers. GMR-const performs much worse.

Table 7.5: The **test set** classification accuracies for the CENSUS data for different algorithms.

ALGORITHM	ACCURACY
MLP (80 nodes in hidden layer)	88.2%
GMB-full (4 components; $\epsilon = 0.01$)	86.0%
GMB-pruned (4 components; 55-D; $\epsilon = 0.01$)	85.9%
GMB-diagonal (6 components; $\epsilon = 0.01$)	79.6%
LVQ (55 cells)	67.3%
GMR-prune (1 component; $\epsilon = 10^{-4}$, 35-D)	67.1%
GMR-ridge (1 component; $\epsilon = 10^{-4}$, $\rho = 10^{-5}$)	67.3%
GMR-const (1 component; $\epsilon = 10^{-4}$)	11.1%

We also experimented with using GMR for classifying 9 vowels using data from the CENSUS speech corpus (Cole *et al.* 1994). For this task, there are 70 predictor variables and 9 response variables. We summarize our results with GMR algorithms in table 7.5, where we also show the previous results using GMB algorithms (see chapter 6). We note that all the GMR algorithms obtain a substantially lower accuracy than the GMB algorithms. This may be because using GMR, we are fitting a continuous density function to binary response variables. Among the GMR algorithms, GMR-const again performs much worse than GMR-prune or GMR-ridge.

7.4 Discussion

In this chapter, we described Gaussian mixture models for regression. Using a similar approach to Ghahramani and Jordan (1994*a*, 1994, 1994*b*), we modelled the joint density of the response variables y and predictor variables x as a mixture of Gaussians and derived the least squares regression $E[y \mid x]$; a weighted sum of linear regression functions, where the weights are the probabilities of membership of x to the components of the mixture density of x. Modelling the joint density instead of a prediction function has the advantage that, for any given x, we have a model for the density $p(y \mid x)$ instead of just the mean value of this density. This enables us to generate variance estimates

conditioned on the predictor variables; estimates of the model uncertainty.

We propose two new ways of regularizing the local linear regression function:

- local ridge regression, where we shift the local prediction matrices, inducing bias and possibly reducing model variance, and
- PC pruning, where we prune directions of local prediction matrices which induce the least bias when removed.

We call the unregularized model and the above methods of regularizing them as GMR, GMR-ridge and GMR-prune respectively.

We compared linear regression, GMR algorithms, and a multi layer perceptron (MLP) for three standard regression tasks: predicting the housing prices of Boston, the average sunspots count and a chaotic time series. For the Boston housing data, which has only mild non-linearity, there was no significant difference between the errors obtained by a linear regression, MLP, GMR, GMR-ridge and GMR-prune. For the sunspots data, both the regularized predictors (GMR-ridge and GMR-prune) obtained a much lower error than unregularized GMR. For the Mackey-Glass time series data, GMR and regularized GMR algorithms obtained nearly identical errors, and the MLP obtained nearly half the error obtained by the GMR algorithms. We hypothesize that since the data was only 5 dimensional, there were enough data points to obtain good estimates of the joint density (for the parameters explored) and there was no need to regularize the GMR model. Since the data does not have any noise, the performance of all algorithms can probably be improved by increasing the number of parameters.

We also experimented with using GMR algorithms for classification for two speech classification tasks. The GMR algorithms obtained a lower accuracy than Gaussian mixture classifiers (GMB algorithms; see section 6.1). This may be because, when using GMR, we model the density of discrete binary variables using a continuous density function. An alternative approach proposed by Ghahramani and Jordan (1994*b*) uses a mixture density where for each component of the mixture, the predictor variables are Gaussian distributed and the response variables are multinomial distributed. This

approach might improve results for mixture regression models applied to classification tasks.

Based on our results, we conclude that

- the best technique is problem dependent,
- in general, the Gaussian mixture regression (GMR) algorithms perform comparably to multi-layer perceptrons,
- the diagonal covariance matrices assumption is not valid in general and can lead to very poor prediction (e.g see the performance of GMR-const in section 7.3), and
- regularizing the local linear models seems to be effective when the data is high dimensional, the data dependencies are non-linear and the sample data size is limited. Regularizing the models by bounding the covariance matrices *and* PC pruning seems to work best.

Chapter 8

Conclusions and future work

In this dissertation, we have described local linear models for dimension reduction and Gaussian mixture models for classification and regression. We developed the relation between local linear dimension reduction and maximum likelihood signal extraction for a mixture of Gaussians signal-plus-noise model. Empirical results with speech and image data show that local linear models can be faster and more accurate than global models. In this chapter, we summarize the whole document, present our conclusions and describe directions for future work.

8.1 A synopsis of the dissertation

In this section, we summarize the whole document. We first describe models for dimension reduction and then discuss Gaussian mixture models for classification and regression.

8.1.1 Local linear models for dimension reduction

In high dimensions, training data is sparsely distributed and it is difficult to train models which generalize well from small data sets. One way to combat this "curse of dimensionality" (section 1.2) is to reduce dimension of the data and build models in the low dimensional (feature) space. Dimension reduction algorithms are also useful for feature extraction for classification, exploratory data analysis, and data compression. The goal of dimension reduction algorithms is to obtain *compact* encodings which optimize a criterion for *accuracy* of representation. We focus on unsupervised algorithms whose goal

is to obtain m dimensional encodings of n dimensional data vectors ($m < n$) such that the mean squared error between the original vectors and their reconstructions from m dimensional encodings is minimized.

Principal components analysis (PCA; section 2.1) is a classical technique for dimension reduction. PCA builds a global linear model of the data: an m dimensional hyperplane spanned by the leading eigenvectors of the data covariance matrix. PCA can incur a high error whenever the data variables have non-linear dependencies among them. Five layered auto-associative neural networks (FLNs; section 2.2) can represent non-linear dependencies among data variables to form a *global non-linear* model of the data: a smooth m dimensional (possibly curved) manifold that approximates the spread of the data.

Global models for dimension reduction can be inefficient (inaccurate and hard to fit) when the data lies near complex curved manifolds. When the data has different structure in different regions of the input space, a local linear model can be faster and more accurate. We propose the VQPCA model (section 3.2) which partitions the input space into a set of regions using a vector quantizer (VQ) and builds m dimensional PCA coordinates in each local region. In chapter 3, we described several ways of obtaining the VQ partition for VQPCA.

In chapter 4, we showed that local linear dimension reduction approximates maximum likelihood signal estimation for a signal-plus-noise model, where

- the signal density is a mixture of Gaussians, each with $n - m$ negligibly small eigenvalues,
- the noise density is a spherically symmetric Gaussian and
- the noise variance is much smaller than the signal variances.

We presented experimental results comparing PCA, FLNs, and VQPCA for speech and image dimension reduction (section 5.1) and speech feature extraction (section 5.2). Our results suggest that local linear models (VQPCA) can be more accurate and faster

than a global non-linear model (FLNs). We showed that linear mappings from the low dimensional encodings of speech spectral coefficients obtained by PCA, FLNs and VQPCA can estimate formant frequencies (resonance frequencies of the vocal tract) with a low error. We compared VQ and VQPCA for lossy compression of speech and image data (section 5.3), and found that VQ performs the most accurate compression for any given bit-rate. However, VQPCA performs comparably to a VQ when there is a paucity of training data.

8.1.2 Gaussian mixture models for supervised learning

In chapters 6 and 7, we presented Gaussian mixture models for classification and regression and proposed algorithms for regularizing them. We discussed Gaussian mixture Bayes (GMB) classifiers (section 6.1), which use a mixture of Gaussians model for each class conditional density, and use the EM algorithm (see appendix C) to train the mixture models. Using a winner-take-all approximation, we showed the relation between GMB classifiers and hard clustering based classifiers (section 6.1.1) like the LVQ algorithm (appendix D).

Gaussian mixture models incur the curse of dimensionality for high dimensional data, since the models have $O(n^2)$ parameters. We discussed different ways of regularizing GMB classifiers (section 6.2); diagonal or spherically symmetric covariance approximations, adding a constant diagonal matrix to each covariance matrix in each EM iteration to bound the volume elements (determinants), and pruning those eigen-directions of each Gaussian which induce the least (empirically measured) classification error when removed. We presented results for two speech phoneme classification tasks (section 6.3), comparing GMB classifiers, regularized GMB classifiers and a neural network classifier. Our results suggest that GMB classifiers perform comparably to neural networks and regularization was necessary to prevent singular covariance matrices and over-fitting to the training set. In particular, bounding the covariance matrices by adding a constant diagonal matrix during training seems to work best. The algorithms with the diagonal or spherically symmetric covariance assumptions performed the worst.

We presented Gaussian mixture models for regression and proposed new ways of regularizing them. We presented the Gaussian mixture regression (GMR) algorithm which models the joint density of the inputs and outputs as a mixture of Gaussians, and derive the regression function $E[y \mid x]$ (section 7.2); a local linear function of the inputs. We propose two ways of regularizing the regression function: local ridge regression (section 7.2.1) and principal components pruning (PC pruning; section 7.2.2), where we prune those directions of each local prediction matrix which induce the least additional error when pruned. We presented results comparing GMR, regularized GMR algorithms and a neural network (section 7.3). Our results suggest that the GMR model performed comparably with the neural network model and regularization was effective, especially for high dimensional data when the sample size was limited.

8.2 Conclusions

Based on the experimental results presented in chapters 5, 6, and 7, we conclude that local models or Gaussian mixture models provide effective tools for several statistical data processing applications. In this section, we present our conclusions based on the experimental results for each of these applications.

8.2.1 Dimension reduction, feature extraction, and data compression

In chapter 5, we compared PCA, FLNs and VQPCA for the dimension reduction of speech (vowel) data and image (faces) data, and speech feature extraction. We also compared VQ and VQPCA for the lossy compression of speech and image data. Based on our results, we conclude that

- local linear models for dimension reduction can generate more accurate low dimensional encodings than global models. In our experiments, VQPCA encodings incurred nearly half the error incurred by PCA or FLNs.
- local linear models can be trained much faster than global non-linear models. In our experiments, VQPCA algorithms with a multi-stage or tree structured quantizers

trained more than an order of magnitude faster than the best FLNs (the ones which obtained the least error), while obtaining a lower error than FLNs.

- For the local linear models, clustering using "reconstruction distance" (the distance to the local PCA hyperplane) is more effective than clustering using the Euclidean distance. However, when training data is limited, the two methods seem to be comparable. This may be because when training data is limited, we do not have enough data points to accurately estimate all eigen-directions within each local region. For speech data, VQPCA-Recon obtained nearly 15% lower error than VQPCA-Eucl. For image data, which had very few training vectors, there was not much difference between VQPCA-Eucl and VQPCA-Recon.

- Using multi-stage or tree structured VQs to generate the VQPCA partition is an effective way of reducing the computational complexity relative to an an unconstrained VQ. For speech data, the least error was obtained by VQPCA-R-MS.

- The encoding times of VQPCA algorithms (number of FLOPS required to compute the low dimensional encoding) is higher than that of PCA or FLNs, especially for algorithms using reconstruction distance clustering. However, the decoding is much faster and comparable to PCA. This suggests that VQPCA may not be the method of choice for applications requiring real-time encoding (e.g video conferencing). For applications which can tolerate slow encoding but require real time decoding (e.g multi-media video decoding), VQPCA algorithms are preferable because of their accuracy of encoding, fast training and fast decoding.

- Dimension reduction algorithms can be used for speech *feature extraction*. In section 5.2, we built linear maps from three dimensional encodings of raw speech spectral coefficients obtained by PCA, FLNs and VQPCA to the first three hand-labelled formant frequencies (the resonance frequencies of the vocal tract). We obtained a low error (normalized squared error (5.3) $\mathcal{E}_{norm} < 0.25$) with all three methods suggesting that all the algorithms are encoding formants information in

some form.

- For *data compression* (as opposed to dimension reduction), a VQ can always generate lower distortion encodings than VQPCA for a given bit-rate, provided we have a large training set. VQPCA data compression approximates data compression by a two level tree structured VQ (TSVQ), since we perform an initial quantization into Q cells, and build separate VQs with K cells within each of the first level cells to quantize the local m dimensional encodings. When training data is limited, the constraints imposed by the two level quantization can have a regularizing effect, enabling a VQPCA to generate a lower *test set* distortion than a VQ for the same bit rate.

To summarize, local linear models can be faster and more accurate than global nonlinear models for dimension reduction. All dimension reduction algorithms discussed here (PCA, FLNs, VQPCA) seem to encode speech formants information in low dimensional encodings. For data compression, a VQ (a local constant model) obtains the least distortion for a given bit rate.

8.2.2 Classification and regression

In chapter 6, we presented experimental results comparing Gaussian mixture Bayes (GMB) classifiers, regularized GMB classifiers and a neural network classifier for two speech phoneme classification tasks. In chapter 7, we compared Gaussian mixture regression (GMR), regularized GMR algorithms and a neural network for three benchmark regression tasks. Based on our experimental results, we conclude that

- Gaussian mixture models perform comparably to neural networks for classification and regression.

- Regularizing Gaussian mixture models is necessary to prevent singular covariance matrices during parameter estimation (EM algorithm) and to prevent potential over-parametrization. Among the regularization techniques we investigated,

bounding the covariance matrices by adding a constant diagonal matrix during training was the most effective.

- The diagonal and spherically symmetric covariance matrices assumption seems to be invalid for several data sets, since Gaussian mixture models with these assumptions perform the worst, obtaining high errors for both regression and classification.

- For regression, the best regularization scheme was adding a constant diagonal matrix to each covariance matrix in each iteration of the EM algorithm *in tandem with* PC pruning for each local prediction matrix. For classification, adding a constant diagonal matrix to each covariance matrix in each iteration of the EM algorithm was most effective.

8.3 Directions for future work

In this section, we describe directions for future work suggested by the work presented in the previous chapters. Several questions suggest themselves.

- What is the best way to choose the target dimension m for an n to m dimension reduction of data?

- What is the best way to choose the number of local regions for the local models and the number of components for the Gaussian mixture models?

- Nowlan's (1991) thesis shows that a mixture of Gaussians model is related to VQ clustering under certain assumptions. What is the relation between an EM algorithm for training the mixture model and the GLA for training a VQ?

- For lossy data compression, what is the relation between a tree structured VQ (TSVQ), a TSVQ with transform coding and VQPCA? When is a VQ not effective for data compression?

- For classification, can we modify an EM algorithm to train a GMB model in a *discriminative* way (i.e train using class labels)?

- For classification, can we improve the pruning techniques for regularizing GMB models by pruning combinations of eigen-directions or pruning eigen-directions based on analytic estimates of the classification error induced?

In this section, we elaborate on some of these questions in greater depth.

8.3.1 Choosing the target dimension for dimension reduction

In the first part of this document (chapters 2 through 5), we described several dimension reduction algorithms. In this dissertation, we view dimension reduction as a process in which we are specified a target dimension m and we try to generate m dimensional encodings such that the average reconstruction error is minimized. An alternative viewpoint is to choose a target dimension m based on the data and the specific application. A question arises, "How do we choose m?". We outline a few approaches to answer this question.

- If we are specified a threshold of acceptable reconstruction error, we can reduce dimension to different target dimensions, until the error drops below the specified threshold.

- For PCA and VQPCA, which are related to Gaussian mixture models (see chapter 4), we can use an approach based on *statistical hypothesis testing*. Anderson (1963) derives statistical confidence intervals and hypothesis tests for various hypotheses relating to the trailing eigenvalues of an auto-covariance matrix. For determining m, we can test the hypothesis that the trailing $n - m$ eigenvalues are all equal for every Gaussian component, for different values of m, and pick that m which best validates the hypothesis. This hypothesis corresponds to an additive (spherically symmetric) Gaussian noise model similar to the model described in chapter 4. Another alternative is to test the hypothesis that the largest of the trailing $n - m$ eigenvalues is small enough to be negligible.

- For PCA and VQPCA, if we are specified the number of bits for the low dimensional encoding (the bit-rate for lossy data compression) and the number of local

regions (for VQPCA), we can use local *transform coding* (Gersho & Gray 1992) to automatically select the reduced dimension. Transform coding allocates bits to different PCA coefficients based on the variances (eigenvalues) of the principal components. Thus, there is a natural pruning of some components with a very low variance and we implicitly choose m. Note that with this method, m can be different for different regions.

A related issue is the choice of the number of local regions for local linear models or the number of mixture components for a Gaussian mixture model. For the algorithms presented in this document, we vary the number of components over a range of values and choose the value for which the error (average squared reconstruction error for dimension reduction, the number of mis-classifications for classification and the average squared error for regression) for a validation data set is the least. For classification, a possible extension is to allow the mixture models for different classes to have a different number of components and do the cross validation search over this larger parameter space.

8.3.2 Algorithms for lossy data compression

In section 5.3.2, we described a VQPCA *data compression* algorithm and discussed its relation to a two level tree structured VQ (TSVQ; section 3.1.5). Several questions arise here. What is the exact relationship between a VQPCA data compression model, a TSVQ and a two level tree structured transform coding (Gersho & Gray 1992)? When are any of these techniques more effective than a VQ for rate-distortion performance? A possible extension of the work described in section 5.3 would be an empirical comparison of all the algorithms mentioned above for lossy data compression.

8.3.3 Discriminative training for GMB classifiers

In chapter 6, we described Gaussian mixture Bayes (GMB) classifiers, which build a separate mixture of Gaussians model for each class conditional density, and use Bayes discriminant functions for classifying data vectors. GMB classifiers train the mixture

models for each class separately using an EM algorithm. The classification accuracy can potentially be improved by *discriminative training* using class labelled data to learn the *differences* between the structure of the data for different classes. The proposal is to do a gradient ascent on a discriminative cost function which maximizes the spread between the GMB discriminant functions.

Let x denote a feature vector, $\{\Omega^1, \ldots, \Omega^K\}$ denote K classes, and $\{\delta^1(x), \ldots, \delta^K(x)\}$ denote the GMB discriminant functions (6.3). The vector x is assigned to class Ω^I if

$$\delta^I(x) > \delta^J(x)$$

for all $J \neq I$. GMB classifiers use the EM algorithm to maximize the data likelihood given the mixture model *for each class separately*. Let $< x, L >$ denote a class labelled data point, i.e $x \in \Omega^L$. A discriminative training procedure might maximize the expected value of the cost function,

$$C(x, L) = \delta^L(x) - \sum_{I \neq L} \delta^I(x) \,, \tag{8.1}$$

or other cost functions which attempt to increase the magnitude of the discriminant function for the correct class and decrease it for all other classes. Discriminative cost functions such as the one mentioned above are sometimes used to train hidden Markov models for speech recognition (Rabiner 1989). A direction for future research would be to start from GMB discriminant functions, and use the above objective function, to derive parameter update rules for training the mixture discriminant functions.

8.3.4 The relation between an EM algorithm for training a mixture of Gaussians model and the GLA

In appendix C, we describe an *expectation maximization* (EM) algorithm for training a mixture of Gaussians model. In section 3.1.3, we described the generalized Lloyd algorithm (GLA) for training a VQ model. From Nowlan's (1991) work (see appendix B), we know that VQ clustering approximates a *winner-take-all* mixture of Gaussians model. It would be interesting to develop the relation between a EM algorithm for

training a winner-take-all mixture of Gaussians model and the GLA for training a VQ with a Euclidean distortion measure. Marroquin (1995) briefly discusses the relation between EM and GLA for a mixture of Gaussians model with spherically symmetric covariance matrices.

Both EM and GLA are iterative batch-mode algorithms. An EM algorithm for estimating the parameters of a mixture of Q Gaussians iterates the following two steps.

- For each data point, compute the posterior probabilities of membership to all Gaussian components.
- Re-estimate means, covariance matrices, and mixing proportions based upon new estimates of posterior membership probabilities.

A GLA for estimating the parameters of a VQ with Q cells iterates the following two steps.

- Partition the data into Q sets using the given distortion measure. In other words, assign membership of each data point to one of Q regions.
- Re-estimate means (and covariance matrices, if required by the distortion measure) using the new partition.

There seems to be a natural connection between the above two procedures. For the Euclidean distance measure, we can show that the GLA is a winner-take-all version of the EM algorithm for a mixture model of spherically symmetric Gaussians under certain conditions. Future work here would include deriving the explicit connection and exploring the relation between EM and GLA for other VQ distortion measures.

8.3.5 Improved pruning techniques for GMB classifiers

In section 6.2.2, we proposed a method for regularizing GMB classifiers by pruning those eigen-directions of each discriminant function, which induce the least (empirically

measured) error when pruned. Based on our experiments, pruning eigen-directions regularizes the model, but the selection method for picking directions to prune can be improved.

As we suggested in section 6.2.2, estimating saliency as the validation set error induced by pruning *combinations* of eigen-directions can produce better results than pruning each eigen-direction in isolation. However, this procedure is very compute intensive. Another alternative is to get theoretical estimates of the error induced by pruning an eigen-direction.

8.4 Discussion

In this chapter, we gave a synopsis of the whole dissertation, presented our conclusions and described directions for future work.

In summary, we have shown in this dissertation that local models or Gaussian mixture models are effective tools for dimension reduction, feature extraction, exploratory data analysis, classification and regression. For dimension reduction, the local linear models can be more accurate and much faster than global non-linear models. For classification and regression, the Gaussian mixture models perform comparably with feedforward neural networks.

Bibliography

Ahalt, S. C., Krishnamurthy, A., Cheen, P. & Melton, D. (1990), "Competitive learning algorthms for vector quantization", *Neural Networks* **3**, 277–290.

Ahmad, S. & Tresp, V. (1993), "Some solutions to the missing feature problem in vision", *in* S. Hanson, J. Cowan & C. L. Giles, eds, *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann, San Mateo, California, pp. 393–400.

Aitkin, M., Anderson, D. & Hinde, J. (1981), "Statistical modelling of data on teaching styles (with discussion)", *Journal of the Royal Statistical Society A* **144**, 419–461.

Aitkin, M. & Wilson, G. (1980), "Mixture models, outliers, and the EM algorithm", *Technometrics* **22**, 325–331.

Anderson, T. (1963), "Asymptotic theory for principal component analysis", *Ann. J. Math. Stat.* **34**, 122–148.

Anderson, T. W. (1958), *An introduction to multivariate statistical analysis*, John Wiley and Sons Inc., New York.

Barnes, C. & Frost, R. (1990), "Necessary conditions for the optimality of residual vector quantizers", *in Abstracts of the 1990 IEEE International Symposium on Information Theory*, IEEE, Piscataway, NJ, p. 34.

Basford, K. & McLachlan, G. (1985), "Estimation of allocation rates in a cluster analysis context", *Journal of American Statistical Association* **80**, 286–293.

Baum, L. & Eagon, J. (1967), "An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology", *Bulletin of the American Mathematical Society* **73**(3), 360–363.

Baum, L., Petrie, T., Soules, G. & Weiss, N. (1970), "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains", *The Annals of Mathematical Statistics* **41**(1), 164–171.

Bellman, R. (1961), *Adaptive Control Processes*, Princeton University Press, Princeton, NJ.

Bourlard, H. & Kamp, Y. (1988), "Auto-association by multilayer perceptrons and singular value decomposition", *Biological Cybernetics* **59**, 291–294.

Bregler, C. & Omohundro, S. M. (1994), "Surface learning with applications to lipreading", *in* Cowan, Tesauro & Alspector, eds, *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Mateo, California, pp. 43–50.

Bregler, C. & Omohundro, S. M. (1995), "Nonlinear image interpolation using manifold learning", *in* Tesauro, Touretzky & Leen, eds, *Advances in Neural Information Processing Systems 7*, The MIT Press, Cambridge, Massachusetts, pp. 973–980.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984), *Classification and Regression Trees*, Wadsworth & Brooks, Monterey, California.

Broad, D. J. & Clermont, F. (1989), "Formant estimation by linear transformation of the LPC cepstrum", *The Journal of the Acoustical Society of America* **86**(5), 2013–2017.

Broomhead, D. S. (1991), "Signal processing for nonlinear systems", *in* S. Haykin, ed., *Adaptive Signal Processing, SPIE Proceedings Vol. 1565*, SPIE, pp. 228–243.

Cole, R., Muthusamy, Y. & Fanty, M. (1990), "The ISOLET spoken letter database", *Technical Report* CSE 90-004, Oregon Graduate Institute of Science & Technology.

Cole, R., Novick, D., Burnett, D., Hansen, B., Sutton, S. & Fanty, M. (1994), "Towards automatic collection of the U.S census", *in Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, IEEE, Piscataway, NJ, pp. I/93–I/96.

Cottrell, G. W. (1988), "Principal components analysis of images via back propagation", *in Visual Communications and Image Processing 1988, SPIE Proceedings Vol. 1001*, SPIE, pp. 1070–1077.

Cottrell, G. W. & Metcalfe, J. (1991), "EMPATH: Face, emotion, and gender recognition using holons", *in* R. Lippmann, J. Moody & D. Touretzky, eds, *Advances in Neural Information Processing Systems 3*, Morgan Kauffmann, San Mateo, California, pp. 564–571.

Cottrell, G. W., Munro, P. & Zipser, D. (1987), "Learning internal representations from gray-scale images: an example of extensional programming", *in Proceedings of the Ninth Annual Cognitive Science Society Conference, Seattle, Wa*, pp. 461–473.

Cover, T. M. & Thomas, J. A. (1991), *Elements of Information Theory*, John Wiley and Sons Inc, New York.

Darken, C. & Moody, J. (1991), "Note on learning rate schedules for stochastic optimization", *in Advances in Neural Information Processing Systems 3*, Morgan Kaufmann, San Mateo, California, pp. 832–838.

DeMers, D. & Cottrell, G. (1993), "Non-linear dimensionality reduction", *in* Giles, Hanson & Cowan, eds, *Advances in Neural Information Processing Systems 5*, Morgan Kaufmann, San Mateo, California.

Dempster, A., Laird, N. & Rubin, D. (1977), "Maximum likelihood from incomplete data via the EM algorithm", *Journal of the Royal Statistical Society Series B* **39**, 1–38.

Devijver, P. & Kittler, J. (1982), *Pattern recognition: A statistical approach*, Prentice-Hall, Englewood Cliffs, NJ.

Dillon, W. R. & Goldstein, M. (1984), *Multivariate Analysis Methods and Applications*, John Wiley and Sons, New York.

D.O'Shaughnessy (1987), *Speech Communication*, Addison-Wesley, Redwood City, California.

Draper, N. & Smith, H. (1981), *Applied Regression Analysis, 2nd edition*, John Wiley and Sons, New York.

Duda, R. & Hart, P. (1973), *Pattern Classification and Scene Analysis*, John Wiley and Sons Inc., New York.

Everitt, B. & Hand, D. (1981), *Finite Mixture Distributions*, Chapman and Hall, New York.

Farmer, J. D. & Sidorowich, J. J. (1987), "Predicting chaotic time series", *Physical Review Letters* **59**(8), 845–848.

Fisher, W. & Doddington, G. (1986), "The DARPA speech recognition research database : specification and status", *in Proceedings of the DARPA Speech Recognition Workshop*, Palo Alto, CA, pp. 93–99.

Foldiák, P. (1989), "Adaptive network for optimal linear feature extraction", *in Proceedings of the IJCNN*, IEEE, Piscataway, NJ, pp. I 401–405.

Fraser, A. & Dimitriadis, A. (1993), "Hidden Markov models with mixed states", *in* A. Weigend & N. Gershenfeld, eds, *Predicting the Future and Understanding the Past: a Comparison of Approaches*, Addison-Wesley, Redwood City, California, pp. 265–282.

Friedman, J. (1991), "Multivariate adaptive regression splines", *The Annals of Statistics* **19**, 1–141.

Frost, R., Barnes, C. & Xu, F. (1991), "Design and performance of residual quantizers", *in* J. Storer & J. Reif, eds, *Proceedings of Data Compression Conference*, IEEE Computer Society Press, pp. 129–138.

Fukunaga, K. (1972), *Introduction to Statistical Pattern Recognition*, Academic Press Inc., San Diego, California.

Fukunaga, K. & Olsen, D. R. (1971), "An algorithm for finding intrinsic dimensionality of data", *IEEE Transactions on Computers* **C-20**(2), 176–183.

Funahashi, K. (1989), "On the approximate realization of continuous mappings by neural networks", *Neural Networks* **2**, 183–192.

Funahashi, K. (1990), "On the approximate realization of identity mappings by three-layer neural networks", *Technical Report*, Toyohashi University of Technology, Department of Information and Computer Sciences. Translation of Japanese paper in *Denshi Joho Tsushin Gakkai Ronbunshi*, **J73-A**(1), 1990, pp 139-145.

Ganesalingam, S. & McLachlan, G. (1979), "A case study of two clustering methods based on maximum likelihood", *Statistical Neerlandica* **33**, 81–90.

Gemen, S., Bienenstock, E. & Doursat, R. (1992), "Neural networks and the bias/variance dilemma", *Neural Computation* **4**, 1–58.

G.E.Peterson & H.L.Barney (1952), "Control methods used in a study of the vowels", *The Journal of the Acoustical Society of America* **24**(2), 175–184.

Gersho, A. (1979), "Asymptotically optimal block quantization", *IEEE Transactions on Information Theory* **IT-25**(4), 373–380.

Gersho, A. & Gray, R. M. (1992), *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Norwell, Massachusetts.

Ghahramani, Z. (1994), "Solving inverse problems using an EM approach to density estimation", *in Proceedings of the 1993 Connectionist Models Summer School*, Lawrence Erlbaum Publishers, pp. 316–323.

Ghahramani, Z. & Jordan, M. I. (1994*a*), "Learning from incomplete data", *Technical Report* 108, Center for Biological and Computational Learning; Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology.

Ghahramani, Z. & Jordan, M. I. (1994*b*), "Supervised learning from incomplete data via an EM approach", *in* J. D. Cowan, G. Tesauro & J. Alspector, eds, *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Mateo, California, pp. 120–127.

Gnanadesikan, R. (1977), *Methods for Statistical Data Analysis of Multivariate Observations*, John Wiley and Sons, New York.

Golomb, B., Lawrence, D. & Sejnowski, T. (1991), "Sexnet: A neural network identifies sex from human faces", *in* R. Lippmann, J. Moody & D. Touretzky, eds, *Advances in Neural Information Processing Systems 3*, Morgan Kauffmann, San Mateo, California, pp. 572–577.

Golub, G. H. & van Loan, C. F. (1983), *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland.

Guckenheimer, J. & Holmes, P. (1983), *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Vol. 42 of *Applied Mathematical Sciences*, Springer-Verlag.

Harman, H. H. (1976), *Modern Factor Analysis*, 3rd edn, The University of Chicago Press, Chicago.

Hartman, E. & Keeler, J. D. (1991), "Predicting the future: Advantages of semilocal units", *Neural Computation* **3**(4), 566–578.

Hasselblad, V. (1966), "Estimation of parameters for a mixture of normal distributions", *Technometrics* **8**, 431–444.

Hasselblad, V. (1969), "Estimation of finite mixture of distributions from the exponential family", *Journal of the American Statistical Association* **64**, 1459–1471.

Haykin, S. (1994), *Neural Networks, A Comprehensive Foundation*, Macmillan College Publishing Company, New York.

Heck, L. & Chou, K. (1994), "Gaussian mixture model classifiers for machine monitoring", *in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, IEEE, New York, pp. VI/133–VI/136.

Hediger, T., Passamante, A. & Farrell, M. E. (1990), "Characterizing attractors using local intrinsic dimensions calculated by singular-value decomposition and information-theoretic criteria", *Physical Review A* **41**(10), 5325–5332.

Hermansky, H. (1987), "Perceptual linear predictive (PLP) analysis of speech", *The Journal of the Acoustical Society of America* **4**, 1738–1752.

Hertz, J., Krogh, A. & Palmer, R. G. (1991), *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, California.

Hinton, G., Revon, M. & Dayan, P. (1995), "Recognizing handwritten digits using mixtures of linear models", *in* Tesauro, Touretzky & Leen, eds, *Advances in Neural Information Processing Systems 7*, The MIT Press, Cambridge, Massachusetts, pp. 1015–1022.

Hornik, M., Stinchcombe, M. & White, H. (1989), "Multilayer feedforward networks are universal approximators", *Neural Networks* **2**, 359–368.

Hotelling, H. (1933), "Analysis of a complex of statistical variables into principal components", *Journal of Educational Psychology* **24**, 498–520.

Jacobs, R., Jordan, M., Nowlan, S. & Hinton, G. (1991), "Adaptive mixture of local experts", *Neural Computation* **3**, 79–87.

Jain, A. (1989), *Fundamentals of digital image processing*, Prentice-Hall, Englewood Cliffs, NJ.

Jordan, M. & Jacobs, R. (1994), "Hierarchical mixtures of experts and the EM algorithm", *Neural Computation* **6**(2), 181–214.

Juang, B.-H. & Jr., A. G. (1982), "Multiple stage vector quantization for speech coding", *in Proceeding of the IEEE International Conference on Acoustics and Signal Processing*, IEEE, Piscataway, NJ, pp. 597–600.

Kambhatla, N. & Leen, T. K. (1993), "Fast non-linear dimension reduction", *in 1993 IEEE International Conference on Neural Networks*, IEEE, pp. 1213–1218.

Kambhatla, N. & Leen, T. K. (1994), "Fast non-linear dimension reduction", *in* Cowan, Tesauro & Alspector, eds, *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Mateo, California, pp. 152–159.

Karhunen, J. & Joutsensalo, J. (1995), "Generalizations of principal component analysis, optimization problem, and neural networks", *Neural Networks* **8**(4), 549–562.

Kohonen, T. (1988), *Self-Organization and Associative Memory*, 2nd edn, Springer-Verlag, Berlin.

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

Kruskal, J. (1964), "Multi-dimensional scaling by optimizing goodness of fit to a non-metric hypothesis", *Psychometrika* **29**, 1–27.

Kung, S. Y. & Diamantaras, K. I. (1990), "A neural network learning algorithm for adaptive principal component extraction (APEX)", *in Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*, IEEE, Piscataway, NJ, pp. 861–864.

Lapedes, A. & Farber, R. (1987), "Nonlinear signal processing using neural networks: Prediction and system modelling", *Technical Report* LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, New Mexico.

Lazarsfeld, P. & Henry, N. (1968), *Latent Structure Analysis*, Houghton Mifflin, New York.

Leen, T. K. (1991), "Dynamics of learning in linear feature-discovery networks", *Network : Computation in Neural Systems* **2**, 85–105.

Leen, T. K. (1995), "Regularized local linear models for regression", Unpublished research notes, Oregon Graduate Institute of Science & Technology.

Leen, T. K., Rudnick, M. & Hammerstrom, D. (1990), "Hebbian feature discovery improves classifier efficiency", *in Proceedings of the IJCNN*, IEEE, Piscataway, NJ, pp. I–51 to I–56.

Levin, A. U., Leen, T. K. & Moody, J. E. (1994), "Fast pruning using principal components", *in* J. D. Cowan, G. Tesauro & J. Alspector, eds, *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Mateo, California, pp. 35–42.

Linde, Y., Buzo, A. & Gray, R. M. (1980), "An algorithm for vector quantizer design", *IEEE Transactions on Communications* **COM-28**(1), 84–95.

Lloyd, S. (1982), "Least squares quantization in PCM", *IEEE Transactions on Information Theory* **IT-28**(2), 129–137. Unpublished Bell Laboratories technical note. Portions presented at the Institute of Mathematical Statistics Meeting Atlantic City New Jersey September 1957. Published in March 1982 special issue on quantization of the IEEE Transactions on Information Theory.

MacQueen, J. (1967), "Some methods for classification and analysis of multivariate observations", *in Proceedings of IEEE International Conference on Mathematics, Statistics and Probability*, University of California Press, pp. 281–297.

Marroquin, J. (1995), "Measure fields for function approximation", *IEEE Transactions on Neural Networks* **6**(5), 1081–1090.

McLachlan, G. & Basford, K. (1988), *Mixture Models: Inference and Applications to Clustering*, Marcel Dekker Inc., New York.

Moody, J. & Darken, C. (1988), "Learning with localized receptive fields", *in* D. Touretzky, G. Hinton & T. Sejnowski, eds, *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, San Mateo, California, pp. 133–143.

Moody, J. & Darken, C. J. (1989), "Fast learning in networks of locally-tuned processing units", *Neural Computation* **1**, 281–294.

Moody, J. & Yarvin, N. (1992), "Networks with learned unit response functions", *in* J. Moody, S. Hanson & R. Lippmann, eds, *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann, San Mateo, California, pp. 1048–1055.

Morrison, D. F. (1976), *Multivariate Statistical Methods*, McGraw-Hill, New York.

Namphol, A., Arozullah, M. & Chin, S. (1991), "Higher order data compression with neural networks", *in Proceedings of the IJCNN*, IEEE, Piscataway, NJ, pp. I–55,I–59.

Newcomb, S. (1886), "A generalized theory of the combination of observations so as to obtain the best result", *American Journal of Mathematics* **8**, 343–366.

Nowlan, S. (1991), *Soft Competitive Adaptation: Neural Network Learning Algorithms based on Fitting Statistical Mixtures*, PhD thesis, School of Computer Science, Carnegie Mellon University.

Oja, E. (1983), *Subspace Methods of Pattern Recognition*, John Wiley and Sons Inc., New York.

Oja, E. (1989), "Neural networks, principal components, and subspaces", *International Journal of Neural Systems* **1**, 61–68.

Oja, E. (1991), "Data compression, feature extraction, and autoassociation in feedforward neural networks", *in Artificial Neural Networks*, Elsevier Science Publishers B.V. (North-Holland), pp. 737–745.

Ormoneit, D. & Tresp, V. (1995), "Improved Gaussian mixture density estimates using Bayesian penalty terms and network averaging", *Technical Report* FKI-205-95, Technische Universität München.

Pearson, K. (1894), "Contributions to the mathematical theory of evolution", *Philosophical Transactions of the Royal Society of London A* **185**, 71–110.

Plomp, R., Pols, L. & van de Geer, J. (1967), "Dimensional analysis of vowel spectra", *The Journal of the Acoustical Society of America* **41**(3), 707–712.

Poggio, T. & Girosi, F. (1990), "Networks for approximation and learning", *Proceedings of the IEEE* **78**, 1481–1497.

Pols, L., Tromp, H. & Plomp, R. (1973), "Frequency analysis of Dutch vowels from 50 male speakers", *The Journal of the Acoustical Society of America* **53**(4), 1093–1101.

Pols, L., van der Kamp, L. & Plomp, R. (1969), "Perceptual and physical space of vowel sounds", *The Journal of the Acoustical Society of America* **46**(2), 458–467.

Press, W., Flannery, B., Teukolsky, S. & Vetterling, W. (1987), *Numerical Recipes – the Art of Scientific Computing*, Cambridge University Press, New York.

Priebe, C. & Marchette, D. (1991), "Adaptive mixtures: Recursive nonparametric pattern recognition", *Pattern Recognition* **24**(12), 1197–1209.

Rabiner, L. R. (1989), "A tutorial on hidden Markov models and selected applications in speech recognition", *in* A. Waibel & K.-F. Lee, eds, *Readings in Speech Recognition*, IEEE, Piscataway, NJ, pp. 267–296.

Rubner, J. & Tavan, P. (1989), "A self-organizing network for principal component analysis", *Europhysics Letters* **20**, 693–698.

Rumelhart, D., Hinton, G. & Williams, R. (1986), "Learning internal representations by error propagation", *in* D. Rumelhart, J. McClelland & the PDP Research Group, eds, *Parallel Distributed Processing*, The MIT Press, pp. 318–362.

Sanger, T. (1989), "An optimality principle for unsupervised learning", *in* D. Touretzky, ed., *Advances in Neural Information Processing Systems 1*, Morgan Kauffmann, San Mateo, California.

Seber, G. & Wild, C. (1989), *Nonlinear Regression*, John Wiley and Sons, New York.

Srivastava, M. & Lee, G. (1984), "On the distribution of the correlation coefficient when sampling from a mixture of two bivariate normal densities: robustness and outliers", *Canadian Journal of Statistics* **2**, 119–133.

Stensmo, M. & Sejnowski, T. J. (1995), "A mixture model system for medical and machine diagnosis", *in* G. Tesauro, D. S. Touretzky & T. K. Leen, eds, *Advances in Neural Information Processing Systems 7*, The MIT Press, Cambridge, Massachusetts, pp. 1077–1084.

Titterington, D., Smith, A. & Makov, U. (1985), *Statistical Analysis of Finite Mixture Distributions*, John Wiley and Sons Inc., New York.

Tresp, V., Ahmad, S. & Neuneier, R. (1994), "Training neural networks with deficient data", *in* J. D. Cowan, G. Tesauro & J. Alspector, eds, *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Mateo, California, pp. 128–135.

Usui, S., Nakauchi, S. & Nakano, M. (1991), "Internal color representation acquired by a five-layer neural network", *in* T. Kohonen, K. Makisara, O. Simula & J. Kangas, eds, *Artificial Neural Networks*, Elsevier Science Publishers, North-Holland, pp. 867–872.

Watanabe, S. (1965), "Karhunen-Loeve expansion and factor analysis, theoretical remarks and applications", *in Transactions of the 4th Prague Conference on Information Theory, Statistical Decision Functions and Random Processes*, pp. 645–660.

Wax, M. & Kailath, T. (1985), "Detection of signals by information theoretic criteria", *IEEE Transactions on Acoustics, Speech and Signal Processing* **ASSP-33**(2), 387–392.

Widrow, B. & M. Hoff, Jr. (1960), "Adaptive switching circuits", *in IRE WESCON Convention Record*, pp. 96–104.

Wolfe, J. (1970), "Pattern clustering by multivariate mixture analysis", *Multivariate Behavioural Research* **5**, 329–350.

Yang, J. & Dumont, G. A. (1991), "Classification of acoustic emission signals via Hebbian feature extraction", *in Proceedings of the IJCNN*, IEEE, Piscataway, NJ, pp. I–113 to I–118.

Appendix A

The generalized centroid for "reconstruction distance"

Given a set of data points, $\mathcal{X} = \{x^1, \ldots, x^N\}$, $x^I \in \mathcal{R}^n$ for $I = 1, \ldots, N$, the generalized centroid μ with respect to a distortion measure $d(\cdot, \cdot)$ (see section 3.1.2) is defined as

$$\mu = \operatorname*{argmin}_r \frac{1}{N} \sum_{I=1}^{N} d(x^I, r) \ .$$

Given $\mathcal{X}$, a vector $r \in \mathcal{R}^n$ and orthonormal eigenvectors $\{e_1, \ldots, e_m, e_{m+1}, \ldots, e_n\}$ of the matrix

$$\Sigma(r) = \frac{1}{N} \sum_{I=1}^{N} (x^I - r)(x^I - r)^T \ ,$$

the reconstruction distance is defined (see section 3.2.2) as

$$d(x, r) = (x - r)^T \left(\sum_{i=m+1}^{n} e_i {e_i}^T \right) (x - r) \ .$$

The generalized centroid for reconstruction distance is

$$\mu = \operatorname*{argmin}_r \frac{1}{N} \sum_{I=1}^{N} (x^I - r)^T \left(\sum_{i=m+1}^{n} e_i {e_i}^T \right) (x^I - r) \ . \tag{A.1}$$

Let $C(r)$ denote the right hand side of (A.1), the cost function to be minimized with respect to the vector r,

$$C(r) = \frac{1}{N} \sum_{I=1}^{N} (x^I - r)^T \left(\sum_{i=m+1}^{n} e_i {e_i}^T \right) (x^I - r) \ . \tag{A.2}$$

Taking the derivative of $C(r)$ with respect to the k^{th} component of r, r_k, we obtain,

$$\begin{aligned}
\frac{\partial C(r)}{\partial r_k} &= -\frac{2}{N}\sum_{I=1}^{N}\left(\sum_{i=m+1}^{n} e_i e_i^T\right)(x_k^I - r_k) \\
&\quad + \frac{1}{N}\sum_{I=1}^{N}(x^I - r)^T\left(\sum_{i=m+1}^{n}\frac{\partial e_i}{\partial r_k}e_i^T\right)(x^I - r) \\
&\quad + \frac{1}{N}\sum_{I=1}^{N}(x^I - r)^T\left(\sum_{i=m+1}^{n} e_i\frac{\partial e_i^T}{\partial r_k}\right)(x^I - r) \\
&= -\frac{2}{N}\sum_{I=1}^{N}\left(\sum_{i=m+1}^{n} e_i e_i^T\right)(x_k^I - r_k) \\
&\quad + \frac{2}{N}\sum_{I=1}^{N}(x^I - r)^T\left(\sum_{i=m+1}^{n}\frac{\partial e_i}{\partial r_k}e_i^T\right)(x^I - r) \quad . \qquad \text{(A.3)}
\end{aligned}$$

In section A.1, we show that the derivative $\frac{\partial e_i}{\partial r_k}$ is given by

$$\frac{\partial e_i}{\partial r_k} = -\sum_{j\neq i}^{n}\left\{\frac{e_j^T(\bar{x} - r)e_{ik} + (\bar{x} - r)^T e_i e_{jk}}{\lambda_i - \lambda_j}\right\}e_j \quad , \qquad \text{(A.4)}$$

where $\bar{x} = \frac{1}{N}x^I$ is the arithmetic mean of $\mathcal{X}$. Substituting (A.4) into (A.3), we obtain

$$\begin{aligned}
\frac{\partial C(r)}{\partial r_k} &= -\frac{2}{N}\sum_{I=1}^{N}\left(\sum_{i=m+1}^{n} e_i e_i^T\right)(x_k^I - r_k) \\
&\quad + \frac{2}{N}\sum_{I=1}^{N}\sum_{i=m+1}^{n}\sum_{j\neq i}^{n}\left\{\frac{e_j^T(\bar{x} - r)e_{ik} + (\bar{x} - r)^T e_i e_{jk}}{\lambda_i - \lambda_j}\right\}(x^I - r)^T e_j e_i^T (x^I - r) \\
&= -\frac{2}{N}\sum_{I=1}^{N}\left(\sum_{i=m+1}^{n} e_i e_i^T\right)(x_k^I - r_k) \\
&\quad + 2\sum_{i=m+1}^{n}\sum_{j\neq i}^{n}\left\{\frac{e_j^T(\bar{x} - r)e_{ik} + (\bar{x} - r)^T e_i e_{jk}}{\lambda_i - \lambda_j}\right\} \\
&\quad e_j^T\left\{\frac{1}{N}\sum_{I=1}^{N}(x^I - r)(x^I - r)^T\right\}e_i \qquad \text{(A.5)}
\end{aligned}$$

But, by definition $\Sigma(r) = \frac{1}{N}\sum_{I=1}^{N}(x^I - r)(x^I - r)^T$, and e_i are orthonormal eigenvectors of $\Sigma(r)$. Therefore $e_j^T\Sigma(r)e_i = 0$ if $j \neq i$. Using this, the expression (A.5) reduces to

$$\frac{\partial C(r)}{\partial r_k} = -\frac{2}{N}\sum_{I=1}^{N}\left(\sum_{i=m+1}^{n} e_i e_i^T\right)(x_k^I - r_k)$$

or in vector form

$$\frac{\partial C(r)}{\partial r} = -\frac{2}{N}\sum_{I=1}^{N}\left(\sum_{i=m+1}^{n} e_i e_i^T\right)(x^I - r) \quad . \tag{A.6}$$

Equating the gradient to 0, we get

$$\left(\sum_{i=m+1}^{n} e_i e_i^T\right)\bar{x} = \left(\sum_{i=m+1}^{n} e_i e_i^T\right) r \quad . \tag{A.7}$$

Any vectors r satisfying (A.7) is a generalized centroid for reconstruction distance. We use $\mu = \bar{x}$, the arithmetic mean (which is a solution to (A.7)) as the generalized centroid in section 3.2.2.

A.1 The derivative of e_i with respect to the mean r

We compute the derivative of the eigenvector e_i of the matrix $\Sigma(r) = \frac{1}{N}\sum_{I=1}^{N}(x^I - r)(x^I - r)^T$ with respect to r_k using a first order perturbation expansion of $\Sigma(r)$ and e_i. Perturb r to $\hat{r} = r + \delta r$. The modified matrix $\hat{\Sigma}(r) = \frac{1}{N}\sum_{I=1}^{N}(x^I - \hat{r})(x^I - \hat{r})^T$ can be approximated to first order in δr as

$$\begin{aligned} \hat{\Sigma}(r) &\approx \Sigma(r) + \Delta \\ &\equiv \Sigma(r) - (\bar{x} - r)\delta r^T - \delta r(\bar{x} - r)^T \quad , \end{aligned} \tag{A.8}$$

where $\Delta = -(\bar{x} - r)\delta r^T - \delta r(\bar{x} - r)^T$ is a real symmetric perturbation matrix. The eigenvectors of a perturbed matrix $\hat{\Sigma}(r) = \Sigma(r) + \Delta$ in terms of the eigenvectors of $\Sigma(r)$ to a first order approximation, are given by (e.g see Fukunaga's (1972) book or Golub and van Loan's (1983) book for a derivation)

$$\begin{aligned} \hat{e}_i &\approx e_i + \delta e_i \\ &= e_i + \sum_{j\neq i}^{n} \frac{e_j^T \Delta e_i}{(\lambda_i - \lambda_j)} e_j \\ &= e_i - \sum_{j\neq i}^{n} \frac{e_j^T(\bar{x} - r)\delta r^T e_i + e_j^T \delta r(\bar{x} - r)^T e_i}{(\lambda_i - \lambda_j)} e_j \end{aligned} \tag{A.9}$$

Therefore,

$$\delta e_i = -\sum_{j\neq i}^{n} \frac{e_j^T(\bar{x} - r)\delta r^T e_i + e_j^T \delta r(\bar{x} - r)^T e_i}{(\lambda_i - \lambda_j)} e_j$$

and the derivative $\frac{\partial e_i}{\partial r_k}$ is given by

$$\begin{aligned} \frac{\partial e_i}{\partial r_k} &= \lim_{\delta r_k \to 0} \frac{\delta e_i}{\delta r_k} \\ &= -\sum_{j \neq i}^{n} \frac{e_j^T(\bar{x} - r)e_{ik} + e_i^T(\bar{x} - r)e_{jk}}{(\lambda_i - \lambda_j)} e_j \ . \end{aligned} \tag{A.10}$$

Appendix B

A probabilistic model for VQ clustering

In this appendix, following (Nowlan 1991), we derive the correspondence between vector quantizer (VQ; section 3.1) clustering and a winner-take-all mixture of Gaussians data model. In their classic book on pattern recognition, Duda and Hart (1973) mention this correspondence. They discuss maximum likelihood parameter estimation for a mixture of Gaussians model and describe "a simple approximate procedure" (section 6.4.4 in (Duda & Hart 1973)) which is a generalized Lloyd algorithm (GLA; see section 3.1.3) for VQ clustering using Euclidean distance measure.

In section 3.1.1 of chapter 3, we described VQ clustering and algorithms for training VQs. Recall that a VQ with Q cells partitions the input space into Q disjoint regions and approximates each region with a reference vector. When the VQ distance function ($d(x, \mu)$) is the Euclidean distance, the clustering approximates a maximum likelihood parameter estimation for a mixture of Gaussians data model. Steven Nowlan (1991) derives the precise correspondence between VQ and mixture models in his doctoral thesis. In this chapter, we give a brief summary of his derivation.

When the distance function of a VQ is the Euclidean distance, then the distortion D (see 3.5 in section 3.1) is

$$D = E\left[\min_{k=1}^{Q} \|x - \mu_k\|^2\right] . \tag{B.1}$$

When estimating the parameters $\theta = \{\mu_1, \ldots, \mu_Q\}$ from a training set $\mathcal{X} = \{x^1, \ldots, x^N\}$, the cost function to be minimized is

$$D = \frac{1}{N}\sum_{l=1}^{N}\left[\min_{k=1}^{Q} \|x - \mu_k\|^2\right] , \tag{B.2}$$

which is an estimate of (B.1). The least squares estimates of the parameters are

$$\theta^{*} = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{I=1}^{N} \left[\min_{k=1}^{Q} \|x - \mu_k\|^2 \right] . \tag{B.3}$$

We will show that the least squares estimates θ^{*} are the same as the maximum likelihood estimates for a constrained mixture of Gaussians data model.

We assume that the input density $p(x)$ is a mixture of Q multivariate n-dimensional Gaussians,

$$p(x) = \sum_{k=1}^{Q} \frac{\alpha_k}{(2\pi)^{n/2}\sqrt{|\Sigma_k|}} \exp\left[-\frac{1}{2}(x-\mu_k)^T \Sigma_k{}^{-1} (x-\mu_k)\right] \tag{B.4}$$

where μ_k and Σ_k denote the means and the covariance matrices. We assume that for a given x, the summation in (B.6) can be approximated by the maximum term in the summation,

$$p(x) = \max_{k=1}^{Q} \frac{\alpha_k}{(2\pi)^{n/2}\sqrt{|\Sigma_k|}} \exp\left[-\frac{1}{2}(x-\mu_k)^T \Sigma_k{}^{-1} (x-\mu_k)\right] . \tag{B.5}$$

This assumption, known as the *"winner-take-all"* (WTA) assumption (Nowlan 1991), enables us to relate mixture models and hard clustering algorithms. The WTA assumption assigns all the responsibility for an observation x to the Gaussian which best explains x.

To derive the relation to a VQ clustering, we further assume that

- all the component Gaussians are spherically symmetric (i.e $\Sigma_k = \sigma^2 I_{n\times n}$ for all k).
- all the mixing proportions are equal (i.e $\alpha_k = 1/Q$ for all k).

Using these assumptions, we obtain,

$$p(x) = \max_{k=1}^{Q} Z \exp\left[-\frac{\| x - \mu_k \|^2}{2\sigma^2}\right] , \tag{B.6}$$

where $Z = 1/(Q(2\pi)^{n/2}(\sigma^2)^{n/2})$.

We obtain the maximum likelihood parameter estimates by maximizing the likelihood of a training set $\mathcal{X} = \{x^1, \ldots, x^N\}$. We assume that $x^1, \ldots, x^N$ are independent and identically distributed. The maximum likelihood estimate of the variance σ^2 is

$$\hat{\sigma^2} = \sum_{I=1}^{N} \frac{\min_{k=1}^{Q} \| x - \mu_k \|^2}{Nn} ,$$

where N is the number of vectors in $\mathcal{X}$ and n is the input dimensionality. The maximum likelihood estimate of the parameters $\theta = \{\mu_1, \ldots, \mu_Q\}$ is:

$$\begin{aligned}
\hat{\theta} &= \underset{\theta}{\text{argmax}}\ p(\mathcal{X}) \\
&= \underset{\theta}{\text{argmax}} \prod_{I=1}^{N} p(x^I) \ \text{ using i.i.d,} \\
&= \underset{\theta}{\text{argmax}} \sum_{I=1}^{N} \log(p(x^I)) \ \text{ taking logs,} \\
&= \underset{\theta}{\text{argmax}} \sum_{I=1}^{N} \left[\log(Z) - \frac{\min_{k=1}^{Q} \| x - \mu_k \|^2}{2\sigma^2} \right] \ \text{ using (B.5),} \\
&= \underset{\theta}{\text{argmin}} \sum_{I=1}^{N} \left[\frac{\min_{k=1}^{Q} \| x - \mu_k \|^2}{2\sigma^2} \right] \ \text{ since } Z \text{ is independent of } \theta, \\
&= \underset{\theta}{\text{argmin}} \sum_{I=1}^{N} \left[\min_{k=1}^{Q} \| x - \mu_k \|^2 \right] \ \text{ since } \sigma^2 \text{ is independent of } \theta. \qquad \text{(B.7)}
\end{aligned}$$

The maximum likelihood parameter estimates (B.7) are identical to the least squares estimates (B.3). Thus, a VQ clustering with Euclidean distance is equivalent to maximum likelihood parameter estimation for a winner-take-all mixture of Gaussians model when

- the component Gaussians are spherically symmetric and
- the mixing proportions are equal.

Appendix C

An introduction to the EM algorithm

The *Expectation Maximization (EM)* algorithm is an iterative algorithm for the computation of maximum likelihood parameter estimates when the observations can be viewed as incomplete data. Although the EM algorithm is generally attributed to a 1977 paper by Dempster, Laird and Rubin (1977), different forms of the EM algorithm have been derived in the context of specific problems much earlier (Hasselblad 1966, Hasselblad 1969, Wolfe 1970, Baum & Eagon 1967, Baum, Petrie, Soules & Weiss 1970) (also see (Nowlan 1991), from where these references are reproduced). I am grateful to Professor Andrew Fraser for pointing out the references (Baum & Eagon 1967, Baum *et al.* 1970) to me which form the basis for the Baum-Welch algorithm used for parameter estimation of hidden Markov models (HMMs). In their book on pattern recognition, Duda and Hart (1973) describe an iterative algorithm (section 6.4.3) for maximum likelihood parameter estimation of a mixture of Gaussians model, which is an instance of an EM algorithm. The EM algorithm is widely used for parameter estimation of mixture models, in particular for a mixture of Gaussians model (Dempster *et al.* 1977, Nowlan 1991, Jordan & Jacobs 1994, Ghahramani 1994, Ghahramani & Jordan 1994*b*, Ghahramani & Jordan 1994*a*). The reader is referred to (Dempster *et al.* 1977, Nowlan 1991) for a detailed treatment. Here, we describe the general EM algorithm and a EM algorithm for maximum likelihood parameter estimation of a mixture of Gaussians model.

Suppose we know observed values of a random variable X and we wish to model the density of X using a model parametrized by θ. We wish to obtain parameter estimates $\hat{\theta}$ which maximize the likelihood $\mathcal{L}(\theta) = p(X \mid \theta)$. We assume that this estimation is

intractable and that values of a *missing* or *hidden* random variable Z would make the problem more tractable.

Let $p(X, Z \mid \theta)$ denote the joint probability density of X and Z parametrized by θ. We assume that X and Z are such that maximizing the complete data likelihood $\mathcal{L}_c(\theta) = p(X, Z \mid \theta)$ is more tractable than maximizing $\mathcal{L}(\theta)$. However, we do not know the values of Z. The EM algorithm tackles this problem by iteratively generating a probability distribution over the values of Z and estimating the parameters which maximize the expected value of $\mathcal{L}_c(\theta)$ with respect to Z. The algorithm starts with an initial guess θ_0 of the parameter values. It then repeatedly applies the following two steps to generate successively better parameter estimates $(\theta_1, \theta_2, \ldots)$:

E step: Compute the function

$$Q(\theta \mid \theta_k) = E\left[\mathcal{L}_c(\theta) \mid \mathcal{X}, \theta_k\right] , \tag{C.1}$$

where $E[\cdot]$ denotes an expectation with respect to Z, $\mathcal{X}$ denotes a training set of sample values of X, and θ_k denotes the parameter values after k iterations of the EM algorithm. The function $Q(\theta \mid \theta_k)$ is our estimate of the expected value of the complete data (X and Z) likelihood based on our current best estimate of the model parameters. The phrase "E step" stands for the expectation step.

M step: Compute the parameters which maximize the function $Q(\theta \mid \theta_k)$,

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}}\ Q(\theta \mid \theta_k) . \tag{C.2}$$

The "M step" or the maximization step performs a maximum likelihood estimation of parameters for the expected joint density.

The EM algorithm generates parameter values which maximize (to a local optimum) both $\mathcal{L}(\theta)$ and $\mathcal{L}_c(\theta)$ (Dempster *et al.* 1977). We will now present an EM algorithm for maximum likelihood parameter estimation of a mixture of Gaussians model.

C.1 An EM algorithm for a mixture of Gaussians model

We assume that the probability density function of an n dimensional random vector X is a mixture of Q multivariate Gaussians,

$$p(X = x \mid \theta) = \sum_{j=1}^{Q} \frac{\alpha_j}{(2\pi)^{n/2}\sqrt{|\Sigma_j|}} \exp\left[-\frac{1}{2}(x-\mu_j)^T \Sigma_j{}^{-1} (x-\mu_j)\right] \quad , \tag{C.3}$$

where $\theta = \{\alpha_j, \mu_j, \Sigma_j \mid j = 1, \ldots, Q\}$ is the set of parameters of the model, α_j are the mixing proportions, and μ_j and Σ_j denote the means and the covariance matrices respectively. Suppose we have a sample data set (training set) $\mathcal{X} = \{x^1, \ldots, x^N\}$ of values of X. We assume that the elements of $\mathcal{X}$ are independent and identically distributed (i.i.d) as (C.3). The goal is to obtain parameter values $\hat{\theta}$ which maximise the likelihood of $\mathcal{X}$ given the data, i.e

$$\begin{aligned} \hat{\theta} &= \underset{\theta}{\text{argmax}}\ p(\mathcal{X} \mid \theta) \\ &= \underset{\theta}{\text{argmax}} \prod_{i=1}^{N} p(x^i \mid \theta) \quad , \end{aligned} \tag{C.4}$$

where we have used the i.i.d assumption. We assume a generative model of the data, where each data point $x^i \in \mathcal{X}$ is generated by one and only one of the component Gaussians. The hidden random variable Z for the EM algorithm is the label of the component Gaussian to which X belongs.

We now present an EM algorithm for maximum likelihood parameter estimation for a mixture of Gaussians data model.

- Initialize the means, μ_j, to randomly picked data points from $\mathcal{X}$ and the covariance matrices, Σ_j, to unit matrices. Set $\alpha_j = 1/Q$ for all j. Set the iteration counter $t = 0$.

- **E-step:** Compute the posterior probability $h_{ij}(t) \equiv E[z_{ij} \mid x^i, \theta]$ of membership of x^i to the j^{th} Gaussian, for all i and j,

$$h_{ij}(t) = \frac{\frac{\alpha_j(t)}{(2\pi)^{n/2}\sqrt{|\Sigma_j(t)|}} \exp\left[-\frac{1}{2}(x^i-\mu_j(t))^T \Sigma_j(t)^{-1} (x^i-\mu_j(t))\right]}{\sum_{k=1}^{Q} \frac{\alpha_k(t)}{(2\pi)^{n/2}\sqrt{|\Sigma_k(t)|}} \exp\left[-\frac{1}{2}(x^i-\mu_k(t))^T \Sigma_k(t)^{-1} (x^i-\mu_k(t))\right]} \quad , \tag{C.5}$$

where $\theta(t) = \{\alpha_j(t), \mu_j(t), \Sigma_j(t)\}$ denotes the set of parameters after the tth iteration.

- **M-step:** Re-estimate the mixing proportions, means and covariances of the Gaussians using the data set weighted by $h_{ij}(t)$. First compute the new mixing proportions as

$$\alpha_j(t+1) = \frac{1}{N} \sum_{i=1}^{N} h_{ij}(t) \; .$$

Next, compute the new means as the weighted sum of all the sample data points,

$$\mu_j(t+1) = \frac{\sum_{i=1}^{N} h_{ij}(t) \; x^i}{\sum_{i=1}^{N} h_{ij}(t)} \; .$$

Finally, compute the new covariance matrices as a weighted sum of outer products,

$$\Sigma_j(t+1) = \frac{\sum_{i=1}^{N} h_{ij}(t) \; (x^i - \mu_j(t+1)) \; (x^i - \mu_j(t+1))^T}{\sum_{i=1}^{N} h_{ij}(t)} \; .$$

Increment the iteration counter $t = t + 1$ and go to the E-step.

- Iterate the above two steps until the change in the means is below some specified threshold.

In the above algorithm, a small diagonal matrix is often added to each covariance matrix after each M-step to prevent covariance matrices from becoming singular (Ghahramani 1994, Ormoneit & Tresp 1995). This is also an effective means of regularizing the mixture density, especially when the data size is limited. See Chapter 6 for a discussion of schemes for regularizing Gaussian mixture densities. We have described an EM algorithm for maximum likelihood parameter estimation of a mixture of Gaussians model. Please refer to (Dempster *et al.* 1977, Ghahramani 1994, Ghahramani & Jordan 1994*a*) for a discussion of the theory, applications and properties of the EM algorithm.

Appendix D

The learning vector quantization (LVQ) algorithm

The learning vector quantization (LVQ) (Kohonen 1988) algorithm is a clustering based classification algorithm which adapts the placement of reference vectors (or means) based on a set of class labelled data points. Suppose there are K classes and Q reference vectors. Each of the reference vectors μ_i is labelled with a class S_i. During a training iteration, an input vector x (say of class Ω^I), is randomly drawn from the training set and presented to the quantizer. We compute the closest (Euclidean distance) reference vector $\mu_{w(x)}$ to x. The update rule moves $\mu_{w(x)}$ closer to x if $\mu_{w(x)}$ has the same class label as x and moves $\mu_{w(x)}$ farther from x if it has a different class label from x. Thus, the update rule is

$$\begin{aligned}
\mu_i(t+1) &= \mu_i(t) + \alpha(t)[x(t) - \mu_i(t)] && \text{if } i = w(x) \text{ and } S_i = \Omega^I, \\
\mu_i(t+1) &= \mu_i(t) - \alpha(t)[x(t) - \mu_i(t)] && \text{if } i = w(x) \text{ and } S_i \neq \Omega^I, \\
\mu_i(t+1) &= \mu_i(t) && \text{if } i \neq w(x), \qquad \text{(D.1)}
\end{aligned}$$

where t is the iteration number and $\alpha(t)$ is a learning rate which is annealed as learning progresses. After training is complete, a data point x is assigned to the class Ω^I corresponding to the class label of the reference vector $\mu_{w(x)}$ closest to x. This assignment procedure is equivalent to using the discriminant functions of a winner-take-all Gaussian mixture Bayes (GMB) classifier under certain assumptions (see section 6.1.1 for more details).

Biographical Note

Nanda was born on March 1st, 1969 in a city called Hyderabad in Andhra Pradesh, India. He graduated from his high school, Central School Golconda, Hyderabad in 1986. He then went to college in the *Institute of Technology, Benaras Hindu University* (IT-BHU) in Benaras, Uttar Pradesh, India. He obtained his bachelor's degree (Bachelor of Technology) in computer science and engineering from IT-BHU in 1990. During the summer of 1989, Nanda worked at the Electronics Corporation of India Ltd (ECIL), Hyderabad, designing and implementing a data base management system for the inventory control of the systems testing division of ECIL. After finishing his B.Tech, he came to the U.S to pursue higher studies. He joined the Oregon Graduate Institute in fall 1990 and went on to do a PhD in neural network modelling.