UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

EDIGLEISON FRANCELINO CARVALHO

# Probabilistic Incremental Learning for Image Recognition: Modelling the Density of High-dimensional Data

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Paulo Martins Engel
Advisor

Porto Alegre, February 2014

**CIP – CATALOGING-IN-PUBLICATION**

Carvalho, Edigleison Francelino

Probabilistic Incremental Learning for Image Recognition: Modelling the Density of High-dimensional Data / Edigleison Francelino Carvalho. – Porto Alegre: PPGC da UFRGS, 2014.

64 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2014. Advisor: Paulo Martins Engel.

1. Local projection. 2. Probabilistic learning. 3. Online learning. 4. Incremental learning. 5. High-dimensional data. 6. Gaussian mixture models. 7. Image recognition. I. Engel, Paulo Martins. II. Título.

If I have seen further than others, it is by
standing upon the shoulders of giants.
— Isaac Newton

When it is not in our power to follow what is
true, we ought to follow what is most probable.
— Rene Descartes

# ACKNOWLEDGMENTS

First of all I would like to express my sincere thanks to my advisor, Dr. Paulo Martins Engel, who has provided guidance, valuable comments and support, and for his patience throughout the last two years.

I am very grateful to share the LIAC (Connectionist Artificial Intelligence Laboratory) lab with great people and researchers, and for the long and useful discussions that we had.

I wish to thank the Informatics Institute for providing a highly stimulating research environment and excellent professors who afforded the knowledge necessary to accomplish this work. Thanks also to CAPES for providing financial support.

I also thank my former supervisor Dr. Cheng Li who has helped me to improve my research skills, providing me tips about organization of experiments and giving technical talks during my internship at Bioinformatics Institute (BII), in agency A*STAR, in Singapore.

I would like to thank my family, in special my parents and brothers for their support in my education and in my life, and for their comprehension since we can not actually be in the same place. I thank my friends Dr. Nildo Loiola and his wife, Thaís Jordana, and my mother in law, Aurea Viana, for their friendship and support.

Finally, I thank my wife and best friend Vanessa. She is the least expected treasure I have found or ever expect to find. I cannot imagine having made it this far without her patience, encouragement and loyalty.

# CONTENTS

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| **EM** | Expectation-Maximization |
| **L-BFGS** | Limited Memory Broyden-Fletcher-Goldfarb-Shanno |
| **IGMN** | Incremental Gaussian Mixture Network |
| **LP-IGMN** | Local Projection Incremental Gaussian Mixture Network |
| **MLP** | Multilayer Perceptron |
| **NLDD** | Non-linear discriminative dictionary |
| **PCA** | Principal Component Analysis |
| **KPCA** | Kernel Principal Component Analysis |
| **LDA** | Linear Discriminant Analysis |
| **PSNR** | Peak Signal to Noise Ratio |

# LIST OF FIGURES

# LIST OF TABLES

# ABSTRACT

Nowadays several sensory systems provide data in flows and these measured observations are frequently high-dimensional, i.e., the number of measured variables is large, and the observations are arriving in a sequence. This is in particular the case of robot vision systems. Unsupervised and supervised learning with such data streams is challenging, because the algorithm should be capable of learning from each observation and then discard it before considering the next one, but several methods require the whole dataset in order to estimate their parameters and, therefore, are not suitable for online learning. Furthermore, many approaches suffer with the so called *curse of dimensionality* (BELLMAN, 1961) and can not handle high-dimensional input data. To overcome the problems described above, this work proposes a new probabilistic and incremental neural network model, called Local Projection Incremental Gaussian Mixture Network (LP-IGMN), which is capable to perform life-long learning with high-dimensional data, i.e., it can continuously learn considering the stability of the current model's parameters and automatically adjust its topology taking into account the subspace's boundary found by each hidden neuron. The proposed method can find the intrinsic subspace where the data lie, which is called the *principal subspace*. Orthogonal to the principal subspace, there are the dimensions that are noisy or carry little information, i.e., with small variance, and they are described by a single estimated parameter. Therefore, LP-IGMN is robust to different sources of data and can deal with large number of noise and/or irrelevant variables in the measured data. To evaluate LP-IGMN we conducted several experiments using simulated and real datasets. We also demonstrated several applications of our method in image recognition tasks. The results have shown that the LP-IGMN performance is competitive, and usually superior, with other state-of-the-art approaches, and it can be successfully used in applications that require life-long learning in high-dimensional spaces.

**Keywords:** Local projection, Probabilistic learning, Online learning, Incremental learning, High-dimensional data, Gaussian mixture models, Image recognition.

# RESUMO

Atualmente diversos sistemas sensoriais fornecem dados em fluxos e essas observações medidas são frequentemente de alta dimensionalidade, ou seja, o número de variáveis medidas é grande, e as observações chegam em sequência. Este é, em particular, o caso de sistemas de visão em robôs. Aprendizagem supervisionada e não-supervisionada com esses fluxos de dados é um desafio, porque o algoritmo deve ser capaz de aprender com cada observação e depois descartá-la antes de considerar a próxima, mas diversos métodos requerem todo o conjunto de dados a fim de estimar seus parâmetros e, portanto, não são adequados para aprendizagem em tempo real. Além disso, muitas abordagens sofrem com a denominada maldição da dimensionalidade (BELLMAN, 1961) e não conseguem lidar com dados de entrada de alta dimensionalidade. Para superar os problemas descritos anteriormente, este trabalho propõe um novo modelo de rede neural probabilístico e incremental, denominado Local Projection Incremental Gaussian Mixture Network (LP-IGMN), que é capaz de realizar aprendizagem perpétua com dados de alta dimensionalidade, ou seja, ele pode aprender continuamente considerando a estabilidade dos parâmetros do modelo atual e automaticamente ajustar sua topologia levando em conta a fronteira do subespaço encontrado por cada neurônio oculto. O método proposto pode encontrar o subespaço intrísico onde os dados se localizam, o qual é denominado de subespaço principal. Ortogonal ao subespaço principal, existem as dimensões que são ruidosas ou que carregam pouca informação, ou seja, com pouca variância, e elas são descritas por um único parâmetro estimado. Portanto, LP-IGMN é robusta a diferentes fontes de dados e pode lidar com grande número de variáveis ruidosas e/ou irrelevantes nos dados medidos. Para avaliar a LP-IGMN nós realizamos diversos experimentos usando conjunto de dados simulados e reais. Demonstramos ainda diversas aplicações do nosso método em tarefas de reconhecimento de imagens. Os resultados mostraram que o desempenho da LP-IGMN é competitivo, e geralmente superior, com outras abordagens do estado da arte, e que ela pode ser utilizada com sucesso em aplicações que requerem aprendizagem perpétua em espaços de alta dimensionalidade.

**Palavras-chave:** Projeção local, Aprendizagem probabilística, Aprendizagem perpétua, Aprendizagem incremental, Dados de alta dimensionalidade, Modelos de mistura de Gaussianas, Reconhecimento de imagens.

# 1 INTRODUCTION

In unsupervised learning problems, the objective may be partition the data into homogeneous groups, where it is called *clustering*, or to determine the distribution of data within the input space, known as *density estimation* (BISHOP, 2006). In particular, unsupervised learning in high-dimensional spaces is a problem in many fields, including image analysis. The images can be represented by high-dimensional vectors, where each variable is the intensity of each pixel, and learning with these images can be very challenging for many algorithms. These algorithms suffer from the well-known *curse of dimensionality* (BELLMAN, 1961).

However, it has been known that real data will often be confined to a region of the space having lower effective dimensionality, and in particular the directions over which important variations in the target variables occur may be so confined (BISHOP, 2006). It is possible to develop successful methods able to work with high-dimensional data if we can exploit this property and find the subspace where most variation on data occurs. This is the case of well-known approaches such as Principal Component Analysis (PCA) (JOLLIFFE, 2005) which is a dimensionality reduction method that finds a compact representation of data through a linear transformation of the original space to a lower dimensional subspace, then this new representation of the input vector can be used to train an algorithm. A drawback of PCA and several dimensionality reduction methods is that they need the whole dataset to estimate their parameters and, therefore, they are not suitable for online settings. In online learning the data points are arriving in a sequence (data streams) and we need to learn from each data point and then discard it before considering the next point. This type of learning requires low storage resources and is common applied in robotic tasks (SAIDI et al., 2007), and it is our focus is this work.

Models which estimate the density of the input data are widely used because their probabilistic foundations and their advantage in obtaining partitions of data that can be interpreted from a statistical point of view. This is the case of the Incremental Gaussian Mixture Network (IGMN) (HEINEN, 2011)(ENGEL; HEINEN, 2010a)(ENGEL; HEINEN, 2010b) which is an online and incremental model-based algorithm capable to perform supervised and unsupervised learning. Unlike most artificial neural networks such as the Multilayer Perceptron (MLP) (RUMELHART; HINTON; WILLIAMS, 1986) which has its topology (number of hidden layers and neurons) predefined by the user and can not adapt itself using only the input data, IGMN creates new neurons, dependent on the complexity of the current task, in order to model the input data density correctly. The ability to continuously acquire new knowledge considering an incremental topology and the stability of the already learned knowledge makes IGMN a life-long (KIRSTEIN, 2010) learning algorithm.

However, IGMN shows a disappointing behaviour in high-dimensional spaces which is mainly due to the fact that it is over-parametrized in these spaces.

In this work we propose a new neural network model, called Local Projection Incremental Gaussian Mixture Network (LP-IGMN), which is able to perform probabilistic and life-long learning in high-dimensional spaces without suffering with the curse of dimensionality, overcoming the IGMN problems described above. The main advantages of LP-IGMN over other approaches are:

- It takes into account the specific subspace around which each neuron is located and therefore limits the number of parameters to estimate;
- It models the data density (using Gaussian units) and therefore the learned model is not a "black box" and can be analysed using statistical tools;
- LP-IGMN can be used as a visualization tool of high-dimensional data;
- The LP-IGMN just needs one scan over the training data to learn and each data point can be immediately used and discarded, therefore, it requires low storage resources;
- LP-IGMN is a life-long learning algorithm and therefore does not have a separate phase for learning and recalling, i.e., it can continuously acquire new knowledge without forget the previous learned knowledge;
- It adjusts its topology incrementally considering the subspace's boundary of each neuron and the stability of the already learned knowledge;
- It has few non critical configuration parameters that are easy to configure;
- The LP-IGMN has a computational complexity that is linear in the number of data points being suitable for online learning.
- LP-IGMN can be used in supervised and unsupervised learning tasks in high-dimensional spaces with redundant and irrelevant input dimensions.

LP-IGMN is also particularly useful in image recognition tasks as we will show later in chapter 4. In particular, we will demonstrate that LP-IGMN is an excellent performer in face, handwritten digits and object recognition tasks and it can be applied in image denoising and segmentation. This is because the input image vectors have high dimensionality, but the important information lie in a subspace of lower dimensionality and LP-IGMN can properly find it, even when the input data are highly nonlinear.

## 1.1 Main Contributions

This Section summarizes the main contributions of this work. The main objective of this dissertation is to develop a new approach for life-long learning of high-dimensional data streams. In particular, we want to be able of modelling effectively the density of high-dimensional input data and to perform unsupervised and supervised learning in image recognition tasks.

The first contribution of this work is a new neural network model, called LP-IGMN. This algorithm can be seen as an extension of the IGMN algorithm to deal with high-dimensional data streams. IGMN is not capable of properly handle these type of data and fails while is learning. LP-IGMN overcomes this problem and offers interesting features for data analysis such as its capability of locally projecting the high-dimensional data onto a 2-dimensional subspace where most variation on data occurs, giving us an insight about the data. Furthermore, we will show that LP-IGMN is a highly successful learning algorithm and an excellent performer on many standard benchmarks and is also useful for practical applications.

The second contribution of this work is the use of LP-IGMN in many machine learning and image recognition tasks. In fact, the efficiency of LP-IGMN opens new possibilities and research directions where relevant developments can be made. One of these directions is in learning complex image features hierarchies using our proposed model embedded in a deep learning pipeline (BENGIO, 2009; HINTON; SALAKHUTDINOV, 2006).

## 1.2 Outline

This dissertation will proceed as follows:

**Chapter 2: Background.** We review some state-of-the-art approaches to deal with high-dimensional data and also the artificial neural networks used in this work. These algorithms are used later in comparison with LP-IGMN, where we point out the advantages of our proposed method over these standard approaches.

**Chapter 3: Modelling High-dimensional Data Streams with the Local Projection Incremental Gaussian Mixture Network.** We present the new neural network model proposed in this dissertation, called LP-IGMN, which is the main contribution of this work. Throughout that chapter the mathematical derivation of LP-IGMN is presented as well as its new model parameters and its learning and recalling algorithm. We also describe several analyses performed with real and simulated high-dimensional data to evaluate the LP-IGMN performance and to show its main features.

**Chapter 4: Applications for Image Recognition and Representation.** Many applications of the proposed method are described. Specifically, we show how LP-IGMN can be applied for face, handwritten digits and object recognition, image denoising and segmentation. We will also compare our approach with other state-of-the-art algorithms and find that, indeed, LP-IGMN performance is competitive, and usually superior, to more sophisticated systems.

Finally, **Chapter 5** concludes this monograph summarizing the main concepts and contributions of this dissertation and suggesting directions for further work.

# 2 BACKGROUND

Since the dimension of observed data is usually higher than their intrinsic dimension, it is theoretically possible to reduce the dimension of the original space without losing relevant information. For this reason, dimension reduction methods are frequently used in practice to reduce the dimension of the data before the clustering or classification step and to deal with the curse of dimensionality. In next Section, we briefly review some of these techniques that we will refer throughout this dissertation. Another approach to handle with the curse of dimensionality is to consider it as a problem of over-parametrized modelling. For instance, the number of parameters in Gaussian models increases with the square of the dimensionality and this yields inference problems with high-dimensional data. BOUVEYRON; GIRARD; SCHMID (2007) proposed a parametrization of the Gaussian mixture model which yields a family of 28 parsimonious models. To this end, they re-parametrize the covariance matrices into the group specific eigenspaces and constrain model parameters within or across those eigenspaces.

In next Sections, we also describe the artificial neural networks used throughout the experiments performed in next chapters. For instance, IGMN is the core algorithm which we intend to extend to deal with high-dimensional data.

This chapter is organized as follows: Section 2.1 provides an overview of the main dimensionality reduction techniques used in the literature and Section 2.2 describes the artificial neural networks used throughout this dissertation.

## 2.1 Dimensionality Reduction

### 2.1.1 Principal Component Analysis

Principal Component Analysis (PCA) (JOLLIFFE, 2005) is a standard technique for dimensionality reduction which finds a subspace spanned by a set of basis vectors that correspond to the maximum-variance directions in the original space. PCA learns a linear transformation matrix $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_M) \in \mathbb{R}^{D \times M}$ that maps the original $D$-dimensional data point $\mathbf{x}_n$ onto a lower dimensional representation $\mathbf{z}_n = \mathbf{W}^T \mathbf{x}_n \in \mathbb{R}^M$, $n = 1, ..., N$, where usually $M \ll D$. We shall assume that the value of $M$ is given and that $\mathbf{x}_n$ has zero mean. We can also retroproject the data onto the original space using the inverse transformation $\tilde{\mathbf{x}}_n = \mathbf{W}\mathbf{z}_n$. The matrix $\mathbf{W}$ contains the $M$ eigenvectors $\mathbf{w}_1, ..., \mathbf{w}_M$ of the data covariance matrix $\mathbf{\Sigma} = (1/N) \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T$ corresponding to the $M$ largest eigenvalues $\lambda_1, ..., \lambda_M$.

### 2.1.2 Incremental PCA

The Incremental PCA (HALL; MARSHALL; MARTIN, 1998) is a version of PCA to deal with data streams. It estimates a set of parameters $\Theta^{(n+1)}$ by successively updating an earlier state $\Theta^{(n)}$ as new observations become available. The parameters $\Theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{W})$ consist of the mean vector $\boldsymbol{\mu}$, covariance matrix $\boldsymbol{\Sigma}$ and the orthonormal eigenvectors matrix $\mathbf{W}$ which spans a $M$-dimensional subspace of maximum variance of the data. Consider a data flow of $N$ data points $\mathbf{x}_n \in \mathbb{R}^D$, $n = 1, ..., N$. When a new observation $\mathbf{x}_{n+1}$ is presented to the model, the mean vector is updated as:

$$\boldsymbol{\mu}^{(n+1)} = \frac{1}{N+1}(N\boldsymbol{\mu}^{(n)} + \mathbf{x}_{n+1}) \tag{2.1}$$

and the covariance matrix as:

$$\boldsymbol{\Sigma}^{(n+1)} = \frac{N}{N+1}\boldsymbol{\Sigma}^{(n)} + \frac{N}{(N+1)^2}(\mathbf{x}_{n+1} - \boldsymbol{\mu}^{(n)})(\mathbf{x}_{n+1} - \boldsymbol{\mu}^{(n)})^T. \tag{2.2}$$

The new eigenvectors matrix $\mathbf{W}^{(n+1)}$ is calculated as a rotation $\mathbf{R}$ of the current eigenvectors $\mathbf{W}^{(n)}$ plus an orthonormal vector. The additional vector is a unit residue vector $\tilde{\mathbf{r}} = \frac{\mathbf{r}}{||\mathbf{r}||_2}$ for $||\mathbf{r}||_2 \neq 0$ and $\tilde{\mathbf{r}} = 0$ otherwise, where $\mathbf{r} = (\mathbf{x}_{n+1} - \boldsymbol{\mu}^{(n)}) - \mathbf{W}\mathbf{W}^T(\mathbf{x}_{n+1} - \boldsymbol{\mu}^{(n)})$. The new eigenvectors matrix $\mathbf{W}^{(n+1)} \in \mathbb{R}^{D\times(M+1)}$ is computed by:

$$\mathbf{W}^{(n+1)} = \left[\mathbf{W}^{(n)}, \tilde{\mathbf{r}}\right]\mathbf{R}, \tag{2.3}$$

where $\mathbf{R} \in \mathbb{R}^{(M+1)\times(M+1)}$ is a rotation matrix. $\mathbf{R}$ yields from the solution of the following eigenproblem:

$$\left[\mathbf{W}^{(n)}, \tilde{\mathbf{r}}\right]^T \boldsymbol{\Sigma}^{(n+1)} \left[\mathbf{W}^{(n)}, \tilde{\mathbf{r}}\right]\mathbf{R} = \mathbf{R}\boldsymbol{\Lambda}. \tag{2.4}$$

Note that the new eigenvectors matrix $\mathbf{W}^{(n+1)}$ has $(M+1)$ eigenvectors, therefore, it spans a subspace of $(M+1)$ dimensions. If we want to keep the subspace dimension as $M$, we need to discard one eigenvector. A simple approach is only to keep the $M$ eigenvectors associated to the $M$ largest eigenvalues in $\boldsymbol{\Lambda}$.

### 2.1.3 Kernel Principal Component Analysis

Kernel Principal Component Analysis (KPCA) (SCHöLKOPF; SMOLA; MüLLER, 1998) is a nonlinear kernel-based extension of PCA. This algorithm first maps the $D$-dimensional data into a high-dimensional feature space $\mathcal{F}$ of dimension $M$ (which can be infinite dimensional) via a (usually nonlinear) function $\phi(.) : \mathbb{R}^D \to \mathbb{R}^M$ and then performs linear PCA on the mapped data.

Given a set of $N$ data points $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n, ..., \mathbf{x}_N\}$, $\mathbf{x}_n \in \mathbb{R}^D$, KPCA finds directions in which the projected variables $\mathbf{w}^T\phi(\mathbf{x}_n)$, $n = 1, ..., N$ have maximal variance. Assuming zero-mean-mapped data $\sum_{n=1}^N \phi(\mathbf{x}_n) = 0$, the covariance matrix in the feature space becomes $\mathbf{C} = (1/N)\sum_{i=1}^N \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T$ with eigendecomposition $\mathbf{C}\mathbf{w} = \lambda\mathbf{w}$. As the feature space $\mathcal{F}$ might be very high dimensional, the estimation of $\phi(.)$ and $\mathbf{C}$ is complicated. KPCA employs Mercer kernels instead of carrying out the mapping $\phi(.)$ explicitly (SCHöLKOPF; SMOLA; MüLLER, 1998). A Mercer kernel is a function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ that corresponds to a dot product in a high-dimensional feature space $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$. A widely used kernel function is the Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/c)$, where $c$ is a constant.

The KPCA problem can be solved by means of the eigendecomposition of the kernel matrix $\mathbf{K} \in \mathbb{R}^{N \times N}$ which contains pairwise evaluations of the kernel function $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $i, j = 1, ..., N$. Typically, the data are preprocessed in the feature space by removing the mean. This step is accomplished by centering the kernel matrix: $\mathbf{K}_c = \mathbf{K} - \mathbf{1}_N\mathbf{K} - \mathbf{K}\mathbf{1}_N + \mathbf{1}_N\mathbf{K}\mathbf{1}_N$, where $\mathbf{1}_N$ denotes the $N \times N$ matrix in which every element takes the value $1/N$. Thus we can evaluate $\mathbf{K}_c$ using only the kernel function and then use $\mathbf{K}_c$ to determine the eigenvalues and eigenvectors. Note that the standard PCA algorithm is recovered as a special case if we use a linear kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$.

### 2.1.4 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a method used to find a linear combination of features which characterizes or separates two or more classes and it can be utilized to reduce the dimensionality of the input data (MCLACHLAN, 2004). Let $T = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n), ...., (\mathbf{x}_N, y_N)\}$, $\mathbf{x}_n \in \mathbb{R}^D, y_n \in Y = \{1, 2, ..., C\}$ be a labelled training set with $C$ classes. The within-class covariance matrix to the case of $C$ classes is given by:

$$\mathbf{S}_W = \sum_{c=1}^{C} \sum_{n \in Y_c} (\mathbf{x}_n - \boldsymbol{\mu}_c)(\mathbf{x}_n - \boldsymbol{\mu}_c)^T, \tag{2.5}$$

where $\boldsymbol{\mu}_c$ is the mean of class $c$ and is given by:

$$\boldsymbol{\mu}_c = \frac{1}{N_c} \sum_{n \in Y_c} \mathbf{x}_n \tag{2.6}$$

and $N_c$ is the number of data points in class $Y_c$. The between-class covariance matrix is defined as:

$$\mathbf{S}_B = \sum_{c=1}^{C} N_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T, \tag{2.7}$$

where $\boldsymbol{\mu}$ is the overall mean:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n. \tag{2.8}$$

The goal of the LDA is to train the linear data projection:

$$\mathbf{z} = \mathbf{W}^T\mathbf{x}, \tag{2.9}$$

such that the class separability criterion (DUDA; HART; STORK, 2001):

$$J(\mathbf{W}) = \frac{det(\mathbf{S}_B)}{det(\mathbf{S}_W)} \tag{2.10}$$

is maximized. In other words, LDA aims to maximize the between-class distance while minimizing the within-class distance. Maximization of such criteria is straightforward. The weight values are determined by those eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$ that correspond to the $M$ largest eigenvalues (FUKUNAGA, 1990). It should be noted that there are at most $(C-1)$ nonzero eigenvalues and, therefore, the upper bound on the subspace dimension is $(C-1)$ (i.e., $M \leq C-1$).

## 2.2 Artificial Neural Networks

### 2.2.1 Multilayer Perceptron

The Multilayer Perceptron (MLP) (RUMELHART; HINTON; WILLIAMS, 1986) is a supervised feedforward artificial neural network composed of an input, an output layer and an arbitrary number of hidden layers. In general MLPs are multi-layered and fully connected networks composed of non-linear neurons which propagate signals through the hidden layers until the output layer. The output of each neuron is calculated as the scalar product between its inputs and its incoming weights followed by an activation function $\varphi(.)$, usually a sigmoid function.

The back-propagation algorithm is used to train MLP networks. The error in the output layer, which is computed as the difference between the output and the desired output, is back-propagated through the different hidden layers in order to update the network weights. The weight update of neuron $k$ connected to neuron $l$ in the previous layer is calculated in the following way:

$$\Delta w_{lk}^{(n)} = \eta \delta_k^{(n)} y_l^{(n)}, \tag{2.11}$$

where $\eta$ is the learning rate, $y_l^{(n)}$ is the output of neuron $l$ for the input pattern $\mathbf{x}_n$ and $\delta_k^{(n)}$ is the local gradient defined as:

$$\delta_k^{(n)} = \begin{cases} \varphi'(y_k^{(n)})(t_o^{(n)} - y_o^{(n)}) & : \quad \text{for output neurons} \\ \varphi'(y_k^{(n)}) \sum_j \delta_j^{(n)} w_{kj} & : \quad \text{for hidden neurons} \end{cases}, \tag{2.12}$$

where $j$ are all the neurons that get input from node $k$, $t_o^{(n)}$ is the desired network output for the $n$-th input pattern and output neuron $o$ and $\varphi'(.)$ is the first derivative of the activation function $\varphi(.)$.

MLP networks are considered to be universal function approximators (HORNIK, 1991; MAXWELL; WHITE, 1989), given sufficient number of neurons, and can be utilized for regression and classification tasks. If only few training patterns are available and they change continuously, the network can not maintain the stability of learned knowledge, therefore this kind of network is not suitable for online learning tasks (KIRSTEIN, 2010). Furthermore the back-propagation algorithm is known to converge very slowly (CHRISTIAN; LEBIERE, 1990), requiring many epochs of training in order to find a global minimum of the cost function.

### 2.2.2 Incremental Gaussian Mixture Network

The Incremental Gaussian Mixture Network (IGMN) (HEINEN, 2011)(ENGEL; HEINEN, 2010a)(ENGEL; HEINEN, 2010b) is a probabilistic neural network able to model the distribution of data streams by a mixture of Gaussian components represented by its neurons. It can perform supervised and unsupervised learning in an online way, which means that only a single data point is used at a time to estimate the model parameters. The learning process is fast and "one-shot", meaning that only a single scan through the data is necessary to obtain a consistent model, and can immediately recognize trained data. The IGMN can also adjust its topology incrementally adding or removing neurons as necessary, handling the stability-plasticity dilemma. Furthermore, the IGMN is suitable for life-long learning since it has the ability to continuously acquire new knowledge and considering the stability of already learned knowledge.

The network operation can be summarized in two modes: (a) **learning mode**, in which IGMN updates the neurons for new input patterns if at least one neuron can properly represent the new information, or creates new neurons otherwise, and removes noisy neurons, which represent noise data. The IGMN can perform the learning mode online and perpetually, without suffering from catastrophic interference. Thus, a complete retraining is not necessary when new training data is presented to the network. After at least one learning step, the network can perform the (b) **recalling mode** to estimate the missing elements at the input layer.

The network architecture is composed by two layers: the input layer or visible layer represented by the input patterns and the hidden layer represented by the hidden neurons. The neurons are fully connected across the layers and the connections are bidirectional and have no weights as can be seen in Figure 2.1. There is no need for explicit weights on the connections since all necessary information is wrapped in Gaussian components. The second layer adjusts its size to fit to the training data and both layers work differently when the network is learning or estimating the missing elements at the input layer (recalling). When the network is learning, the input layer only receives the input data and sends them to the hidden layer, without any computation. In IGMN, there is no explicit output layer, any neuron at the input layer can be used as input or output by omitting its value (i.e., presenting an incomplete pattern to the network) on the recalling mode. In this case, the neurons whose input values are known, are used as input elements and their values are sent to the hidden layer, then the neurons whose input values are unknown or missing, receive an estimate of their values from the hidden layer.

The hidden layer is responsible to hold and handle all information of the network. In the learning mode, the hidden layer receives the input pattern from the input layer and assimilates this information by updating the current set of neurons or creating new ones. At the same time, the layer can remove the neurons which are considered noise (i.e., which do not have a minimum activation during a certain number of steps). This process is incremental and "one-shot", i.e., there is no need to present the same information more than once to the network, the pattern is learned in a single step, and it can be performed in an online way. In the recalling mode, the hidden layer receives a partial information from the input layer and uses it to estimate the missing information.

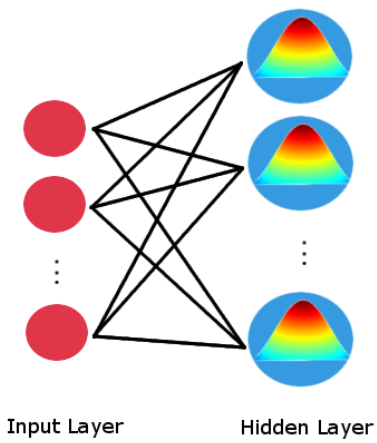



Figure 2.1: IGMN architecture.

### Learning mode

In classification or regression tasks, the IGMN learns a joint distribution $p(\mathbf{x}_a, \mathbf{x}_b)$ between an observed feature vector $\mathbf{x}_a$ (i.e., the independent variables) and a target vector $\mathbf{x}_b$. For clustering tasks, we just suppress the target value from the input vector. In general case, we have an observed data vector $\mathbf{x} \in \mathbb{R}^D$, where $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)^T$, i.e., $\mathbf{x}$ is the concatenation of the vectors $\mathbf{x}_a$ and $\mathbf{x}_b$. Typically, the target value assumes a real value in regression tasks and a binary vector in classification tasks. For classification purpose, we define the target vector as a binary vector of size $m$, where $m$ is the number of classes, and we set the $j$-th position to one and the others to zero, where $j$ is the class label index for the observed feature vector $\mathbf{x}_a$.

Let $\mathbf{\Theta} = (\Theta_1, ..., \Theta_k, ..., \Theta_K)^T$ be the set of parameters in the hidden layer with $K$ neurons, where $\Theta_k = (\boldsymbol{\mu}_k, \mathbf{\Sigma}_k, \pi_k, sp_k, v_k)$ corresponds to the mean vector $\boldsymbol{\mu}_k \in \mathbb{R}^D$, covariance matrix $\mathbf{\Sigma}_k \in \mathbb{R}^{D \times D}$, prior probability or mixing coefficient $\pi_k \in \mathbb{R}$, posterior probability accumulator $sp_k \in \mathbb{R}$ and age $v_k \in \mathbb{N}$ for the $k$-th neuron, with $k = 1, ..., K$. Suppose we have a sequence of $N$ data vectors, $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n, ..., \mathbf{x}_N\}$, the learning process in IGMN updates the current $\mathbf{\Theta}$ after a presentation of $\mathbf{x}_n$, drawn independently from $\mathbf{X}$, in order to maximize the likelihood function:

$$\ln p(X|\mathbf{\Theta}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k)\pi_k \right\}, \tag{2.13}$$

where $\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k)$ is a multivariate Gaussian distribution represented by the $k$-th neuron and takes the form:

$$\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k) = \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{\Sigma}_k|}} \exp\left\{ -\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right\} \tag{2.14}$$

and the prior probability satisfy:

$$0 \leq \pi_k \leq 1, \quad \forall k, \tag{2.15}$$

together with

$$\sum_{k=1}^{K} \pi_k = 1. \tag{2.16}$$

### Creating neurons

In IGMN a new neuron is added when there is no neuron in the hidden layer or when the observed data vector $\mathbf{x}$ matches the minimum likelihood criterion:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k) < \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{\Sigma}_k|}} \exp\left\{ -\frac{1}{2}\chi^2_{D,(1-\beta)} \right\}, \qquad \forall k, \tag{2.17}$$

where $\chi^2_{D,(1-\beta)}$ is the $(1-\beta)$ quantile of the chi-squared distribution with $D$ degrees of freedom and $\beta$ is a user defined parameter that specifies the probability that $\mathbf{x}$ does not belong to the neuron's Gaussian distribution. Since the Mahalanobis distance has a chi-squared distribution (JOHNSON; WICHERN, 2007), i.e., $(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \sim \chi^2_D$, (2.17) can be rewritten to:

$$(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) > \chi^2_{D,(1-\beta)}, \qquad \forall k. \tag{2.18}$$

This new formulation prevents numeric errors when $\mathbf{x}$ is a high dimensional data vector. When the minimum likelihood criterion is reached, a new neuron $K$ is created centered on $\mathbf{x}$, i.e.:

$$\begin{aligned} K &= K+1, \qquad sp_K = 1, \qquad v_K = 0, \\ \boldsymbol{\mu}_K &= \mathbf{x}, \qquad \boldsymbol{\Sigma}_K = \boldsymbol{\sigma}^2_{ini}, \qquad \pi_K = \frac{sp_K}{\sum_{q=1}^{K} sp_q}, \end{aligned} \tag{2.19}$$

where $\boldsymbol{\sigma}^2_{ini} = \text{diag}(\sigma_1^2, ..., \sigma_D^2)$ and $\sigma_1^2, ..., \sigma_D^2$ are an user defined fraction $\delta$ of the overall variance (e.g., $\delta = 0.1$) of the corresponding input variables, estimated from the range of these values according to:

$$\sigma_i^2 = \left(\delta\left[\max(\mathbf{X}_i) - \min(\mathbf{X}_i)\right]\right)^2, \quad \text{for } i = 1, ..., D, \tag{2.20}$$

where $[\max(\mathbf{X}_i) - \min(\mathbf{X}_i)]$ defines the approximate domain of the $i$-th variable.

After the creation of a neuron, all the prior probabilities are adjusted to satisfy constraints (2.15 and 2.16) by:

$$\pi_k = \frac{sp_k}{\sum_{q=1}^{K} sp_q}, \quad \forall k. \tag{2.21}$$

### Updating neurons

The IGMN assumes that the density of the input $\mathbf{x}$ can be modelled by a linear combination of multivariate Gaussian components in the form:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\pi_k. \tag{2.22}$$

Thus the marginal distribution of $\mathbf{x}$ is a Gaussian mixture of the form (2.22). Therefore, the set of parameters in the hidden layer, $\boldsymbol{\Theta}$, can be estimated by an online version of the Expectation-Maximization (EM) algorithm (HEINEN, 2011). The EM algorithm alternates between the following two steps that we shall call the E step and the M step. In the expectation step, or E step, we use the current values for the parameters $\boldsymbol{\Theta}^{(n)}$, in the $n$-th iteration, to evaluate the posterior probabilities, given by:

$$p(k|\mathbf{x}) = \frac{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k^{(n)}, \boldsymbol{\Sigma}_k^{(n)})\pi_k^{(n)}}{\sum_{q=1}^{K} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_q^{(n)}, \boldsymbol{\Sigma}_q^{(n)})\pi_q^{(n)}}, \qquad \forall k. \tag{2.23}$$

We then use these probabilities in the maximization step, or M step, to re-estimate the set of parameters $\Theta_k^{(n+1)}$, for every neuron $k = 1, ..., K$ using:

$$sp_k^{(n+1)} = sp_k^{(n)} + p(k|\mathbf{x}), \tag{2.24}$$

$$v_k^{(n+1)} = v_k^{(n)} + 1, \tag{2.25}$$

$$\boldsymbol{\mu}_k^{(n+1)} = \boldsymbol{\mu}_k^{(n)} + \frac{p(k|\mathbf{x})}{sp_k^{(n+1)}} \left( \mathbf{x} - \boldsymbol{\mu}_k^{(n)} \right), \tag{2.26}$$

$$\begin{aligned} \boldsymbol{\Sigma}_k^{(n+1)} = \boldsymbol{\Sigma}_k^{(n)} - (\boldsymbol{\mu}_k^{(n+1)} - \boldsymbol{\mu}_k^{(n)})(\boldsymbol{\mu}_k^{(n+1)} - \boldsymbol{\mu}_k^{(n)})^T + \\ \frac{p(k|\mathbf{x})}{sp_k^{(n+1)}} \left[ (\mathbf{x} - \boldsymbol{\mu}_k^{(n+1)})(\mathbf{x} - \boldsymbol{\mu}_k^{(n+1)})^T - \boldsymbol{\Sigma}_k^{(n)} \right]. \end{aligned} \tag{2.27}$$

Then all the prior probabilities are adjusted using (2.21) and the updated posterior probability accumulator $sp_k^{(n+1)}$.

### Removing neurons

A neuron $k$ is removed whenever $v_k > v_{min}$ and $sp_k < sp_{min}$, where $v_{min}$ and $sp_{min}$ are manually chosen (e.g., 5.0 and 3.0, respectively). In that case, the prior probabilities must also be adjusted for all remaining neurons $q \in \{1, ..., K\}$, $q \neq k$, using (2.21). In other words, each neuron is given some time ($v_{min}$) to show its importance to the model in the form of an accumulation of its posterior probabilities $sp_k$.

### Recalling mode

Let us recall that the input vector $\mathbf{x}$ can be partitioned into two disjoint subsets $\mathbf{x}_a$ and $\mathbf{x}_b$, i.e.:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix}. \tag{2.28}$$

We also define corresponding partitions of the mean vector $\boldsymbol{\mu}$ given by:

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \tag{2.29}$$

and of the covariance matrix $\boldsymbol{\Sigma}$ given by:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}. \tag{2.30}$$

After at least a single input presentation, the recalling process in IGMN can estimate missing input values from known input values. To simplify our explanation, we will consider that we are estimating a target vector $\hat{\mathbf{x}}_b$ from $\mathbf{x}_a$. In IGMN the conditional distribution $p(\mathbf{x}_b|\mathbf{x}_a)$ can be expressed as:

$$p(\mathbf{x}_b|\mathbf{x}_a) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{b|a}, \boldsymbol{\Sigma}_{b|a}), \tag{2.31}$$

where $\boldsymbol{\mu}_{b|a}$ is given by:

$$\boldsymbol{\mu}_{b|a} = \boldsymbol{\mu}_b + \boldsymbol{\Sigma}_{ba}\boldsymbol{\Sigma}_{aa}^{-1}(\mathbf{x}_a - \boldsymbol{\mu}_a), \tag{2.32}$$

and $\boldsymbol{\Sigma}_{b|a}$ is given by:

$$\boldsymbol{\Sigma}_{b|a} = \boldsymbol{\Sigma}_{bb} - \boldsymbol{\Sigma}_{ba}\boldsymbol{\Sigma}_{aa}^{-1}\boldsymbol{\Sigma}_{ab}. \tag{2.33}$$

This represents a *linear-Gaussian* model (BISHOP, 2006). During the recalling mode, each hidden neuron can be viewed as a *linear-Gaussian* model and we can

perform a probabilistic combination of these linear regressors to estimate $\hat{\mathbf{x}}_b$ as (HEINEN, 2011; GHAHRAMANI; JORDAN, 1994):

$$\hat{\mathbf{x}}_b = \sum_{k=1}^{K} p(k|\mathbf{x}_a) \left[\boldsymbol{\mu}_{b,k} + \boldsymbol{\Sigma}_{ba,k}\boldsymbol{\Sigma}_{aa,k}^{-1}(\mathbf{x}_a - \boldsymbol{\mu}_{a,k})\right], \tag{2.34}$$

where $p(k|\mathbf{x}_a)$ is given by:

$$p(k|\mathbf{x}_a) = \frac{\mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_{a,k}, \boldsymbol{\Sigma}_{aa,k})\pi_k}{\sum_{q=1}^{K} \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_{a,q}, \boldsymbol{\Sigma}_{aa,q})\pi_q}, \qquad \forall k. \tag{2.35}$$

Let us recall that in classification tasks, the estimated target vector $\hat{\mathbf{x}}_b$ has size $m$, where $m$ is the number of classes, i.e., $\hat{\mathbf{x}}_b = (\hat{x}_{b,1}, ..., \hat{x}_{b,j}, ..., \hat{x}_{b,m})$. In this case, the predicted label $\hat{l}$ is given by the index of the highest response in $\hat{\mathbf{x}}_b$, i.e.:

$$\hat{l} = \arg\max_j \hat{\mathbf{x}}_{b,j}. \tag{2.36}$$

## 2.3 Summary

In this chapter we reviewed several state-of-the-art approaches to deal with high-dimensional data and also the artificial neural networks used in this work. The main drawback of PCA, KPCA and LDA is that they require the entire dataset to estimate a linear transformation matrix which projects the high-dimensional data in a low-dimensional representation. These methods are unsupervised, so if we intend to perform classification, we need to train a classifier in a separate phase after the reduction dimensionality step. Therefore, they can not be applied in online settings. The Incremental PCA treats the problem of estimating the eigenvectors matrix for each observation, being suitable for online learning, but it can only learn a single subspace of the data which usually limits its representation capacity. We also reviewed the classical MLP algorithm and the life-long learning algorithm called IGMN. MLP is commonly used in classification and regression tasks because its capacity to approximate complex non-linear functions. However, MLP is a batch algorithm and its topology can not be adjusted automatically using the input data. IGMN can learn with data flows and adjust its topology automatically to model the data density, but it shows a disappointing behaviour when trained with high-dimensional data. Its covariances matrices become singular in these spaces and fail to invert during the learning process.

In next chapter we will describe an extension of the algorithm IGMN which is capable to find specific subspaces for each group in the data, in contrast to the classical approaches such as PCA, Incremental PCA, KPCA and LDA, and this usually yields a more flexible model that better fits the data density and also obtains better performance results.

# 3 MODELLING HIGH-DIMENSIONAL DATA STREAMS WITH THE LOCAL PROJECTION INCREMENTAL GAUSSIAN MIXTURE NETWORK

This chapter describes the operation of the proposed model in this work, called Local Projection Incremental Gaussian Mixture Network (LP-IGMN), which is an extension for the IGMN algorithm to deal with high-dimensional data streams. IGMN requires to estimate full covariance matrices and therefore the number of parameters increases with the square of the dimension. However, because many dimensions are noisy or carry redudant information it can be assumed that high-dimensional data live around subspaces with a dimensionality lower than the one of the original space (BOUVEYRON; GIRARD; SCHMID, 2007). In LP-IGMN, we introduce local projections in low dimensional subspaces together with a probabilistic version of the incremental PCA (HALL; MARSHALL; MARTIN, 1998) in order to deal with such data and to limit the number of parameters to estimate. We evaluated the performance of LP-IGMN using artificial and real datasets, and the results showed that the algorithm is robust to such data and suitable for online learning.

This chapter is structured as follows: Section 3.1 describes our proposed LP-IGMN algorithm. Section 3.2 details several experiments that we conducted to evaluate our model's parameters and performance, and Section 3.3 presents a summary.

## 3.1 Local Projection Incremental Gaussian Mixture Network

The LP-IGMN assumes that the density of the input pattern $\mathbf{x} \in \mathbb{R}^D$ can be modeled by a linear combination of multivariate Gaussian components represented by its $K$ hidden neurons, considering the orthogonal projection of the data onto a lower dimensional linear space, known as the *principal subspace*, $\xi$, such that the variance of the projected data is maximized. The process of orthogonal projection is illustrated in Figure 3.1.

In LP-IGMN, each hidden neuron $k$ assumes that the conditional density is Gaussian $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ with mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. Consider the orthonormal eigenvectors matrix $\mathbf{U}_k = (\mathbf{u}_{k1}, ..., \mathbf{u}_{ki}, ..., \mathbf{u}_{kD}) \in \mathbb{R}^{D \times D}$ of $\boldsymbol{\Sigma}_k$. The projected covariance matrix, $\boldsymbol{\Lambda}_k$, is a diagonal matrix which contains the eigenvalues of $\boldsymbol{\Sigma}_k$ and it has the following form:

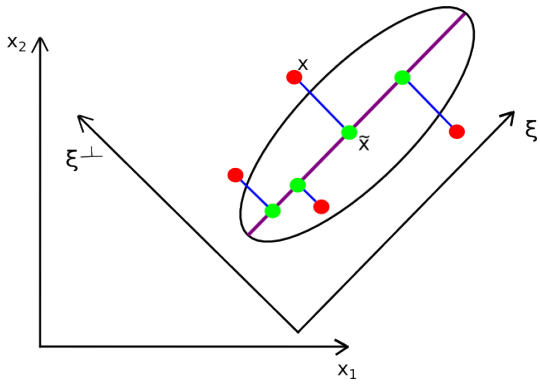



Figure 3.1: Local projection of data points (red dots) onto a lower-dimensional subspace $\xi$ (denoted by the magenta line) found by a LP-IGMN hidden neuron, such that the projection maximizes the variance of the projected points (green dots).

$$\mathbf{\Lambda}_k = \mathbf{U}_k^T \mathbf{\Sigma}_k \mathbf{U}_k = \begin{pmatrix} \begin{matrix} \lambda_{k1} & & 0 \\ & \ddots & \\ 0 & & \lambda_{kd} \end{matrix} & \mathbf{0} \\ \mathbf{0} & \begin{matrix} \lambda_{kd+1} & & 0 \\ & \ddots & \\ 0 & & \lambda_{kD} \end{matrix} \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} d \\ \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} D-d \end{matrix} \tag{3.1}$$

with $\lambda_{k1} \geq \ldots \geq \lambda_{kd} \geq \ldots \geq \lambda_{kD}$, $k = 1, ..., K$ and $d \in \{1, ..., D-1\}$ is an user defined parameter, which determines the number of eigenvectors to keep. We restrict the model fixing the first $d$ eigenvalues to be common within each neuron component, i.e., we assume that $\mathbf{\Lambda}_k$ contains only two different eigenvalues, $a_k$ and $b_k$, where $a_k$ is given by:

$$a_k = \frac{1}{d} \sum_{i=1}^{d} \lambda_{ki}, \tag{3.2}$$

and $b_k$ is given by:

$$b_k = \frac{1}{D-d} \sum_{i=d+1}^{D} \lambda_{ki}. \tag{3.3}$$

It has been shown that adding this constraint is an efficient way to regularize the estimation of $\mathbf{\Lambda}_k$ (BOUVEYRON; GIRARD; SCHMID, 2007). Given the projected covariance matrix $\mathbf{\Lambda}_k$, we can also retrieve the covariance matrix in the original space using the expression $\mathbf{\Sigma}_k = \sum_{i=1}^{D} \lambda_{ki} \mathbf{u}_{ki} \mathbf{u}_{ki}^T$ (BISHOP, 2006).

Similarly to PCA (JOLLIFFE, 2005), LP-IGMN approximates a data point using a representation involving a restricted number $d < D$ of variables corresponding to a projection onto a lower-dimensional subspace. Let $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n, ..., \mathbf{x}_N\} \in \mathbb{R}^{D \times N}$ be a stream of $N$ data points. The $d$-dimensional linear subspace found by a hidden neuron $k$ can be represented, without loss of generality, by the first $d$ eigenvectors in $\mathbf{U}_k$, and so we approximate each data point $\mathbf{x}_n$ by (BISHOP, 2006):

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{d} (\mathbf{x}_n^T \mathbf{u}_{ki}) \mathbf{u}_{ki} + \sum_{i=d+1}^{D} (\boldsymbol{\mu}_k^T \mathbf{u}_{ki}) \mathbf{u}_{ki} \tag{3.4}$$

Since the mean vector $\boldsymbol{\mu}_k$ can be written as:

$$\boldsymbol{\mu}_k = \sum_{i=1}^{D} (\boldsymbol{\mu}_k^T \mathbf{u}_{ki}) \mathbf{u}_{ki} \tag{3.5}$$

we can rewrite (3.4) as:

$$\tilde{\mathbf{x}}_n = \boldsymbol{\mu}_k + \sum_{i=1}^{d} (\mathbf{x}_n^T \mathbf{u}_{ki} - \boldsymbol{\mu}_k^T \mathbf{u}_{ki}) \mathbf{u}_{ki}. \tag{3.6}$$

In some situations, we need a global approximation of the data point instead of a local approximation, which we define as a weighted projection:

$$\tilde{\mathbf{x}}_n^G = \sum_{k=1}^{K} p(k|\mathbf{x}_n) \left\{ \boldsymbol{\mu}_k + \sum_{i=1}^{d} (\mathbf{x}_n^T \mathbf{u}_{ki} - \boldsymbol{\mu}_k^T \mathbf{u}_{ki}) \mathbf{u}_{ki} \right\}, \tag{3.7}$$

where $p(k|\mathbf{x}_n)$ is the posterior probability that $\mathbf{x}_n$ has been generated from the $k$-th neuron Gaussian component and the superscript $G$ indicates the global approximation of $\mathbf{x}_n$. Therefore, we assume that each hidden unit contributes partially to the the input vector global approximation $\tilde{\mathbf{x}}_n^G$.

We define the cost function used to measure the distortion introduced by the reduction in dimensionality as a weighted average of the squared distance between the original data point $\mathbf{x}_n$ and its approximation $\tilde{\mathbf{x}}_n$, so that we want to minimize:

$$J = \sum_{k=1}^{K} \sum_{n=1}^{N} \frac{p(k|\mathbf{x}_n) \, ||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2}{\sum\limits_{n=1}^{N} p(k|\mathbf{x}_n)}. \tag{3.8}$$

Take into account that $\mathbf{x}_n$ can be expressed by a linear combination of the eigenvectors of a neuron as:

$$\mathbf{x}_n = \sum_{i=1}^{D} (\mathbf{x}_n^T \mathbf{u}_{ki}) \mathbf{u}_{ki}. \tag{3.9}$$

Using (3.4) and (3.9) we obtain the expression for the projection error vector as following:

$$\mathbf{x}_n - \tilde{\mathbf{x}}_n = \sum_{i=d+1}^{D} \{(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \mathbf{u}_{ki}\} \mathbf{u}_{ki} \tag{3.10}$$

from which we see that the projection error vector lies in the space $\xi_k^{\perp}$, orthogonal to the principal subspace, because it is a linear combination of $\mathbf{u}_{ki}$ for $i = d+1, ..., D$, as illustrated in Figure 3.1.

We therefore obtain an expression for the distortion measure $J$ as a function of $\mathbf{u}_{ki}$ in the form:

$$J = \sum_{k=1}^{K} \sum_{n=1}^{N} \sum_{i=d+1}^{D} \frac{p(k|\mathbf{x}_n)(\mathbf{x}_n^T \mathbf{u}_{ki} - \boldsymbol{\mu}_k^T \mathbf{u}_{ki})^2}{\sum\limits_{n=1}^{N} p(k|\mathbf{x}_n)} = \sum_{k=1}^{K} \sum_{i=d+1}^{D} \mathbf{u}_{ki}^T \boldsymbol{\Sigma}_k \mathbf{u}_{ki}, \tag{3.11}$$

$$J = \sum_{k=1}^{K} (D-d) b_k, \tag{3.12}$$

thus $J$ is the sum of the eigenvalues of those eigenvectors that are orthogonal to the principal subspace. We therefore obtain the minimum value of $J$ when $b_k \to 0$. LP-IGMN minimizes $J$ by selecting the eigenvectors in the orthogonal subspace $\xi_k^{\perp}$ having the $D-d$ smallest eigenvalues, and hence the eigenvectors defining the principal subspace are those corresponding to the $d$ largest eigenvalues.

### 3.1.1 Creating neurons

Considering the expression $\mathbf{\Sigma}_k^{-1} = \sum_{i=1}^{D} \frac{1}{\lambda_{ki}} \mathbf{u}_{ki}\mathbf{u}_{ki}^T$ (BISHOP, 2006) for the inverse covariance matrix written in the eigenvector space, we can rewrite the Mahalanobis distance $\Delta_k = (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k)$ as:

$$\Delta_k = (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \left\{ \sum_{i=1}^{D} \frac{1}{\lambda_{ki}} \mathbf{u}_{ki}\mathbf{u}_{ki}^T \right\} (\mathbf{x}_n - \boldsymbol{\mu}_k), \tag{3.13}$$

$$\begin{aligned} \Delta_k = (\mathbf{x}_n - \boldsymbol{\mu}_k)^T & \left\{ \sum_{i=1}^{d} \frac{1}{\lambda_{ki}} \mathbf{u}_{ki}\mathbf{u}_{ki}^T \right\} (\mathbf{x}_n - \boldsymbol{\mu}_k) \\ +(\mathbf{x}_n - \boldsymbol{\mu}_k)^T & \left\{ \sum_{i=d+1}^{D} \frac{1}{\lambda_{ki}} \mathbf{u}_{ki}\mathbf{u}_{ki}^T \right\} (\mathbf{x}_n - \boldsymbol{\mu}_k). \end{aligned} \tag{3.14}$$

Since $\mathbf{U}_k$ is an orthonormal matrix, $\mathbf{u}_{ki}^T\mathbf{u}_{ki} = 1$ and using (3.10), we can rewrite (3.14) as:

$$\begin{aligned} \Delta_k = & \sum_{i=1}^{d} \frac{1}{\lambda_{ki}} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \mathbf{u}_{ki} (\mathbf{u}_{ki}^T \mathbf{u}_{ki}) \mathbf{u}_{ki}^T (\mathbf{x}_n - \boldsymbol{\mu}_k) \\ & + \sum_{i=d+1}^{D} \frac{1}{\lambda_{ki}} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \mathbf{u}_{ki} (\mathbf{u}_{ki}^T \mathbf{u}_{ki}) \mathbf{u}_{ki}^T (\mathbf{x}_n - \boldsymbol{\mu}_k). \end{aligned} \tag{3.15}$$

Assuming that the projected covariance matrix is regularized and has only two eigenvalues $a_k$ and $b_k$, we obtain from (3.15):

$$\Delta_k = \frac{1}{a_k} ||\tilde{\mathbf{x}}_n - \boldsymbol{\mu}_k||^2 + \frac{1}{b_k} ||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2 . \tag{3.16}$$

Therefore the Mahalanobis distance in the eigenspace considers the squared distance between the vector approximation $\tilde{\mathbf{x}}_n$ and the mean vector $\boldsymbol{\mu}_k$ weighted by the inverse of the eigenvalue $a_k$ of the principal subspace, and the squared norm of the projection error vector weighted by the inverse of the eigenvalue $b_k$ of the space that is orthogonal to the principal subspace.

Using this formulation for the Mahalanobis distance, we rewrite the IGMN criterion (2.18) to add neurons in the hidden layer as:

$$\frac{1}{a_k} ||\tilde{\mathbf{x}}_n - \boldsymbol{\mu}_k||^2 + \frac{1}{b_k} ||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2 > \chi^2_{D,(1-\beta)}, \qquad \forall k, \tag{3.17}$$

where $\chi^2_{D,(1-\beta)}$ is the $(1-\beta)$ quantile of the chi-squared distribution with $D$ degrees of freedom and $\beta$ is a probability set by the user that controls the sensitivity to component creation. Setting a high value for the parameter $\beta$ increases the sensitivity and more neurons will be added.

Since (3.16) only needs the $d$ eigenvectors associated with the $d$ largest eigenvalues of $\mathbf{\Sigma}_k$ to calculate the data point approximation $\tilde{\mathbf{x}}_n$, LP-IGMN does not need to calculate the $(D-d)$ eigenvectors associated with the smallest eigenvalues whose determination is numerically unstable. Considering the matrix $\mathbf{Q}_k = (\mathbf{u}_{k1}, ..., \mathbf{u}_{ki}, ..., \mathbf{u}_{kd})$ composed by the first $d$ eigenvectors of $\mathbf{U}_k$, the hidden layer in LP-IGMN is fully parametrized by the set of parameters $\mathbf{\Theta} = (\Theta_1, ..., \Theta_k, ..., \Theta_K)^T$, where $\Theta_k = (\boldsymbol{\mu}_k, \mathbf{\Sigma}_k, \pi_k, sp_k, v_k, \mathbf{Q}_k, a_k, b_k)$.

When a data point $\mathbf{x}_n$ matches the criterion (3.17), a new neuron $K$ is added to the LP-IGMN hidden layer and its parameters are initialized as following:

$$\begin{gathered} K = K+1, \qquad sp_K = 1, \qquad v_K = 0, \qquad a_K = a_{ini}, \qquad b_K = b_{ini}, \\ \mathbf{Q}_K = \mathbf{Q}_{ini}, \qquad \boldsymbol{\mu}_K = \mathbf{x}_n, \qquad \mathbf{\Sigma}_K = \boldsymbol{\sigma}^2_{ini}, \qquad \pi_K = \frac{sp_K}{\sum_{q=1}^{K} sp_q}, \end{gathered} \tag{3.18}$$

where $a_{ini} = \frac{1}{d}\sum_{i=1}^{d} \lambda_{ki}$ is the average of the first $d$ largest eigenvalues of $\mathbf{\Sigma}_K = \boldsymbol{\sigma}^2_{ini}$ and $b_{ini} = \frac{1}{D-d}\sum_{i=d+1}^{D} \lambda_{ki}$ is the average of the remaining eigenvalues, i.e., for the $d+1, ..., D$ eigenvalues, and they need to be calculated only once. The eigenvectors matrix $\mathbf{Q}_K = \mathbf{Q}_{ini}$ is initialized with $d$ random orthonormal vectors. For this, one can use the vectors of a matrix $\mathbf{Q}_{ini}$ from a QR decomposition $\mathbf{A} = \mathbf{Q}_{ini}\mathbf{R}$, where the matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ is drawn from the standard normal distribution, i.e., $\mathbf{A} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. After the creation of a neuron, all prior probabilities are adjusted to satisfy constraints (2.15 and 2.16) using (2.21).

### 3.1.2 Expectation Step

As in IGMN, the set of parameters in the hidden layer, $\mathbf{\Theta}$, is estimated by an online version of the EM algorithm. During the E-step or expectation step, we use the current values for the parameters $\mathbf{\Theta}^{(n)}$, in the $n$-th iteration, to evaluate the posterior probabilities $p(k|\mathbf{x}_n)$, $k = 1, ..., K$. Then during the M-step or maximization step, we update the values of the parameters.

Consider the logarithm formulation for the posterior probability:

$$p(k|\mathbf{x}_n) = \frac{\exp\left\{\log \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k^{(n)}, \mathbf{\Sigma}_k^{(n)}) + \log \pi_k^{(n)}\right\}}{\sum_{q=1}^{K} \exp\left\{\log \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_q^{(n)}, \mathbf{\Sigma}_q^{(n)}) + \log \pi_q^{(n)}\right\}}, \tag{3.19}$$

where we obtain $\log \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k^{(n)}, \mathbf{\Sigma}_k^{(n)})$ using (3.16) and the fact that $\log|\mathbf{\Sigma}_k| = d \log a_k^{(n)} + (D-d)\log b_k^{(n)}$:

$$\begin{aligned} \log \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k^{(n)}, \mathbf{\Sigma}_k^{(n)}) = -\frac{1}{2}\Bigg( & \frac{1}{a_k^{(n)}}||\tilde{\mathbf{x}}_n - \boldsymbol{\mu}_k^{(n)}||^2 + \frac{1}{b_k^{(n)}}||\mathbf{x}_n - \tilde{\mathbf{x}}_n||^2 \\ & + d\log a_k^{(n)} + (D-d)\log b_k^{(n)} + D\log 2\pi \Bigg). \end{aligned} \tag{3.20}$$

### 3.1.3 Maximization Step

In order to adapt the LP-IGMN to be suitable for online learning, we developed a probabilistic version of the incremental PCA (HALL; MARSHALL; MARTIN, 1998) to estimate the eigenvectors matrix $\mathbf{Q}_k$ efficiently.

To update the eigenvectors matrix $\mathbf{Q}_k$ incrementally, we need to solve the following eigenproblem:

$$\mathbf{\Sigma}_k^{(n+1)}\mathbf{Q}_k^{(n+1)} = \mathbf{Q}_k^{(n+1)}\mathbf{\Lambda}_k^{(n+1)} \quad \forall k. \tag{3.21}$$

We define that the projection error for a data point $\mathbf{x}_n$ is distributed along the subspaces found by the hidden neurons according to the posterior probability $p(k|\mathbf{x}_n)$, $k = 1, ..., K$, and therefore we calculate the residue vector $\mathbf{r}_k$ in a neuron's subspace as:

$$\mathbf{r}_k = (p(k|\mathbf{x}_n)\mathbf{x}_n - \boldsymbol{\mu}_k) - \sum_{i=1}^{d} \left(p(k|\mathbf{x}_n)\mathbf{x}_n^T\mathbf{u}_{ki} - \boldsymbol{\mu}_k^T\mathbf{u}_{ki}\right)\mathbf{u}_{ki}, \tag{3.22}$$

$$\mathbf{r}_k = \mathbf{y}_n - \tilde{\mathbf{y}}_n, \tag{3.23}$$

where we set $\mathbf{y}_n = p(k|\mathbf{x}_n)\mathbf{x}_n$. In order to obtain the new eigenvectors matrix $\mathbf{Q}_k^{(n+1)}$, we add the unit residue vector $\tilde{\mathbf{r}}_k$ and apply a rotation transformation in the current matrix $\mathbf{Q}_k^{(n)}$:

$$\tilde{\mathbf{r}}_k = \begin{cases} \frac{\mathbf{r}_k}{\|\mathbf{r}_k\|_2}, & \text{if } \|\mathbf{r}_k\|_2 \neq 0, \\ 0, & \text{otherwise,} \end{cases}$$

and therefore we have:

$$\mathbf{Q}_k^{(n+1)} = \left[\mathbf{Q}_k^{(n)}, \tilde{\mathbf{r}}_k\right]\mathbf{R}_k^{(n+1)}, \tag{3.24}$$

where $\mathbf{R}_k^{(n+1)} \in \mathbb{R}^{(d+1)\times(d+1)}$ is a rotation matrix. $\mathbf{R}_k^{(n+1)}$ is the solution of the eigenproblem of the following form:

$$\mathbf{D}_k^{(n+1)}\mathbf{R}_k^{(n+1)} = \mathbf{R}_k^{(n+1)}\mathbf{\Lambda}_k^{(n+1)}, \tag{3.25}$$

where we define $\mathbf{D}_k^{(n+1)} \in \mathbb{R}^{(d+1)\times(d+1)}$ as:

$$\mathbf{D}_k^{(n+1)} = \left[\mathbf{Q}_k^{(n)}, \tilde{\mathbf{r}}_k\right]^T \mathbf{\Sigma}_k^{(n+1)} \left[\mathbf{Q}_k^{(n)}, \tilde{\mathbf{r}}_k\right]. \tag{3.26}$$

Solving (3.25) yields the rotation matrix $\mathbf{R}_k^{(n+1)}$ and the eigenvalues matrix $\mathbf{\Lambda}_k^{(n+1)}$. This intermediate eigenproblem is less complex than an eigenproblem over a full covariance matrix $\mathbf{\Sigma}_k^{(n+1)} \in \mathbb{R}^{D\times D}$ since we usually have $d \ll D$ and the algorithm complexity to solve an eigenproblem with $D$ dimensions is $O(D^3)$. Then the new eigenvectors can be obtained using (3.24). Since $\mathbf{Q}_k^{(n+1)}$ has $d+1$ eigenvectors and we want to keep only $d$ eigenvectors, we select those eigenvectors associated with the $d$ largest eigenvalues in $\mathbf{\Lambda}_k^{(n+1)}$. We then update the variance parameters $a_k$ and $b_k$, for $k = 1, ..., K$:

$$a_k^{(n+1)} = \frac{1}{d}\sum_{i=1}^{d}\lambda_{ki}, \tag{3.27}$$

$$b_k^{(n+1)} = \frac{1}{D-d}\left(Tr(\mathbf{\Sigma}_k^{(n+1)}) - \sum_{i=1}^{d}\lambda_{ki}\right). \tag{3.28}$$

where $\lambda_{ki}$ is the $i$-th largest eigenvalue in $\mathbf{\Lambda}_k^{(n+1)}$ and $Tr(\mathbf{\Sigma}_k^{(n+1)})$ is the trace of the matrix $\mathbf{\Sigma}_k^{(n+1)}$. The other parameters in the LP-IGMN hidden layer are updated as in IGMN, i.e., using (2.21), (2.24), (2.25), (2.26), (2.27). The LP-IGMN learning algorithm is summarized in Algorithm 1.

**Algorithm 1** LP-IGMN Learning

**Init**: $d, \beta, \delta, \boldsymbol{\sigma}^2_{ini}, a_{ini}, b_{ini}, sp_{min}, v_{min}$
  **for all** new data point $\mathbf{x}_n$ **do**:
    Compute the likelihood for all hidden neurons using (3.20).
    **if** $K < 1$ **or** $\frac{1}{a_k}\left|\left|\tilde{\mathbf{x}}_n - \boldsymbol{\mu}_k\right|\right|^2 + \frac{1}{b_k}\left|\left|\mathbf{x}_n - \tilde{\mathbf{x}}_n\right|\right|^2 > \chi^2_{D,(1-\beta)}, \forall k$, **then**
      Create a new hidden neuron and initialize its parameters using (3.18).
    **end if**
    E-step: evaluate the posterior probabilities $p(k|\mathbf{x}_n), \forall k$, using (3.19).
    M-step: update the values of the parameters using (2.21), (2.24), (2.25), (2.26), (2.27), solve the SVD problem (3.25), then update $\mathbf{Q}_k$, $a_k$ and $b_k$ for $k = 1, ..., K$ using (3.24), (3.27) and (3.28).
    **if** $v_k > v_{min}$ **and** $sp_k < sp_{min}$ **then**
      **delete** the $k$-th hidden neuron and adjust the prior probabilities using (2.21).
    **end if**
  **end for**

### 3.1.4 Recalling

During the recalling mode, LP-IGMN works similarly to IGMN, but we need to consider the subspaces found by the hidden neurons. Recall from Section 2.2.2 that the input vector $\mathbf{x}$ can be divided into disjoint vectors $\mathbf{x}_a$ and $\mathbf{x}_b$. In the same way, consider the corresponding partition of the mean vector given by:

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \tag{3.29}$$

and of the covariance matrix given by:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}. \tag{3.30}$$

We can also define the partition of the eigenvector $\mathbf{u}$ given by:

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_a \\ \mathbf{u}_b \end{pmatrix}. \tag{3.31}$$

Note that during the recalling mode, only a partial data vector $\mathbf{x}_a$ is observed and we want to use it to estimate the missing data $\mathbf{x}_b$. First, we need to calculate the posterior probability $p(k|\mathbf{x}_a)$, which is given by:

$$p(k|\mathbf{x}_a) = \frac{\exp\left\{\log \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_{a,k}, \boldsymbol{\Sigma}_{aa,k}) + \log \pi_k\right\}}{\sum_{q=1}^{K} \exp\left\{\log \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_{a,q}, \boldsymbol{\Sigma}_{aa,q}) + \log \pi_q\right\}}, \qquad \forall k. \tag{3.32}$$

In LP-IGMN we calculate the $\log \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_{a,k}, \boldsymbol{\Sigma}_{aa,k})$ as:

$$\begin{aligned} \log \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_{a,k}, \boldsymbol{\Sigma}_{aa,k}) = -\frac{1}{2}\Bigg(&\frac{1}{a_k}\left|\left|\tilde{\mathbf{x}}_a - \boldsymbol{\mu}_{a,k}\right|\right|^2 + \frac{1}{b_k}\left|\left|\mathbf{x}_a - \tilde{\mathbf{x}}_a\right|\right|^2 \\ &+ d\log a_k + (D-d)\log b_k + D\log 2\pi\Bigg), \end{aligned} \tag{3.33}$$

where the approximation $\tilde{\mathbf{x}}_a$ of the data point $\mathbf{x}_a$ is given by:

$$\tilde{\mathbf{x}}_a = \boldsymbol{\mu}_{a,k} + \sum_{i=1}^{d} (\mathbf{x}_a^T \mathbf{u}_{a,ki} - \boldsymbol{\mu}_{a,k}^T \mathbf{u}_{a,ki}) \mathbf{u}_{a,ki}. \tag{3.34}$$

Then recall from (2.34) that the missing vector $\mathbf{x}_b$ can be estimated as:

$$\hat{\mathbf{x}}_b = \sum_{k=1}^{K} p(k|\mathbf{x}_a) \left[ \boldsymbol{\mu}_{b,k} + \boldsymbol{\Sigma}_{ba,k} \boldsymbol{\Sigma}_{aa,k}^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_{a,k}) \right]. \tag{3.35}$$

### 3.1.5 Computation Complexity of LP-IGMN

As each input vector $\mathbf{x}_n$ is processed just once by LP-IGMN, its computational complexity is linear in $N$. However, during the updating it is necessary to perform matrix multiplications, which requires $D^{\log_2 7}$ operations using the Strassen algorithm (STRASSEN, 1969), and is necessary to solve an eigenproblem of a matrix of size $(d+1) \times (d+1)$, which requires $O(d^3)$ computation. Thus, the computational complexity of LP-IGMN is $O(NK(D^{\log_2 7} + d^3))$. It can be noticed that, although LP-IGMN is linear in $N$, the number of operations for each input vector increases as new neurons are added. Since the computational complexity of IGMN is $O(NKD^{\log_2 7})$, the execution time of LP-IGMN is similar to IGMN when $d$ is small ($d \ll D$). This usually happens since we want to keep only a few dimensions that represent the subspace where data live. For large $D$ ($D \gg 10$), the computational cost is still high for both algorithms, and in IGMN this can also result in numeric errors during the inversion of the covariance matrix in (2.14).

## 3.2 Experiments

In this Section, we present and discuss the results of the experiments that we performed on simulated and real data, with the aim of validating the performance of LP-IGMN and of comparing it to other algorithms. The LP-IGMN parameters $\beta, \delta, v_{min}$ and $sp_{min}$ are found using grid search in order to achieve the best result on the test data.

### 3.2.1 Simple 2D Data Simulation

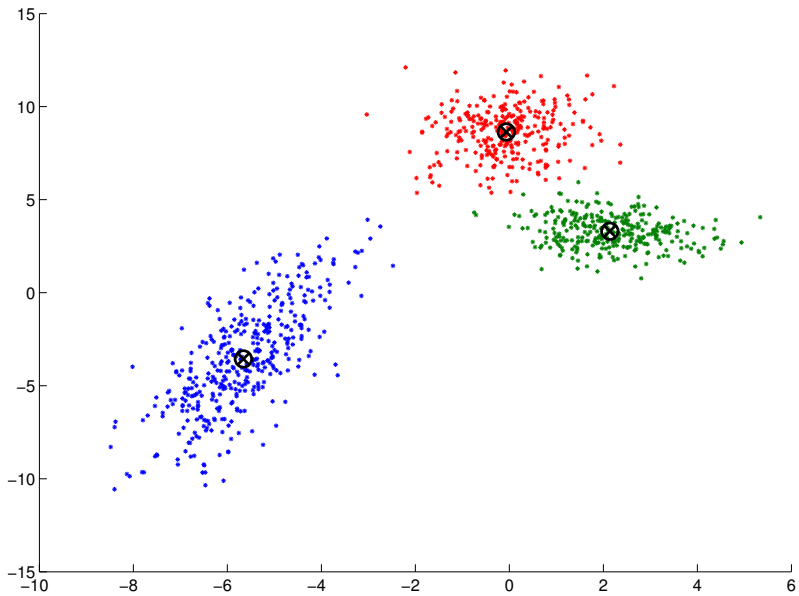
We first perform a simple experiment on a 2D simulated dataset. We have generated a dataset of $N = 1000$ data points $\{\mathbf{x}_1, ..., \mathbf{x}_N\} \in \mathbb{R}^2$ drawn from 3 Gaussian distributions with different means and covariance matrices. The prior probabilities are defined as $\{\pi_1, \pi_2, \pi_3\} = \{0.4, 0.4, 0.3\}$. We perform unsupervised learning using the IGMN and LP-IGMN algorithms and for both we set the following parameters $\beta = 0.1$, $\delta = 0.35$, $v_{min} = 3$, $sp_{min} = 2$. We assume that the data lie on a 1-dimensional subspace and therefore we set $d = 1$ in LP-IGMN. Figure 3.2 shows the results for the assignments of the data points that is obtained by maximum a posteriori rule (MAP), which assigns the data point $\mathbf{x}_n$ to the group $k$ with the highest posterior probability $p(k|\mathbf{x}_n)$. The principal component found by each LP-IGMN hidden neuron is also illustraded in Figure 3.2. The results of both algorithms are similar, this is already expected since the LP-IGMN shows its advantage with more challenging datasets on high dimensional space that we will discuss later in next Sections, however it is interesting that LP-IGMN is capable to

find the 1-dimensional subspace of highest variance and correctly assign the data points to the groups.

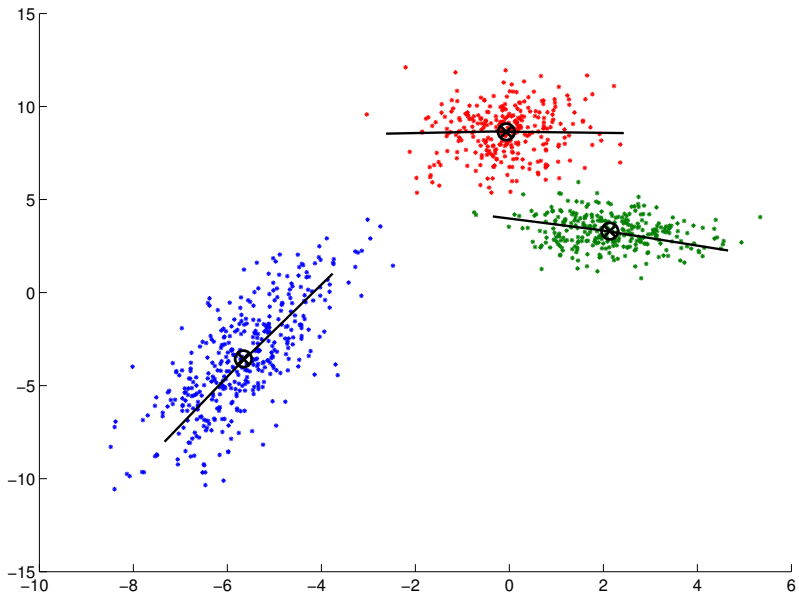
### 3.2.2 Evolution of the Parameters

In this experiment, we are interested in analysing the evolution of the LP-IGMN parameters with the presentation of high-dimensional data and its impact on the accuracy rate. For this, we have generated 3 Gaussian distributions in $\mathbb{R}^{30}$ with the following parameters: $\{d_1, d_2, d_3\} = \{10, 10, 10\}$, $\{\pi_1, \pi_2, \pi_3\} = \{0.4, 0.3, 0.3\}$, $\{a_1, a_2, a_3\} = \{120, 70, 40\}$, $\{b_1, b_2, b_3\} = \{5, 5, 5\}$, close means and random eigenvectors matrices $\mathbf{U}_k$. The parameter $d_1 = 10$ indicates that 10 random dimensions of the first Gaussian distribution have their corresponding eigenvalues set to $a_1$ and the remaining dimensions have their corresponding eigenvalues set to $b_1$. We then generated a stream of $N = 10000$ data points from the above Gaussian distributions where the data points appear in sequence for each group, i.e., the first 4000 vectors in this dataset were drawn from the first Gaussian, the next 3000 vectors were drawn from the second Gaussian and so on. We set the parameters of the LP-IGMN as follows: $\beta = 0.01$, $\delta = 0.3$, $v_{min} = 10$, $sp_{min} = 3$ and we gave the correct value $d$ for the LP-IGMN algorithm. In Figure 3.3 is shown the evolution of the parameters $a_k$ for each neuron in the hidden layer of LP-IGMN. Even with a wrong initialization of the parameters $a_k$ when a new neuron is created, the parameters values converge to the true values when more data arrive.

In order to perform classification using the LP-IGMN, we first generated 11000 data points from the above Gaussian distributions and we separated a random subset of 10000 data points to be used for training and the remaining 1000 to be used for test. The same LP-IGMN parameters above were used and the input vector consisted of a feature vector drawn from a Gaussian distribution concatenated with a binary vector to represent the class label. In this experiment, we simulated a classification problem with 3 classes using each Gaussian distribution above as a different class. We considered that the presentation of 100 data points to the learning process in LP-IGMN is one epoch, and after each epoch of learning, we perform the recalling process using the test data and we calculated the accuracy rate over these data. We compare the results with a restricted version of the IGMN that we called Diag-IGMN, which uses diagonal covariance matrices to deal with high-dimensional data. We do not compare to IGMN with full covariance matrices because this algorithm has not been designed to handle high-dimensional data and may fail due to singularity reasons when attempting to invert its covariance matrices. The parameters in Diag-IGMN were set as the same of LP-IGMN. In Figure 3.4, the results are presented. For the LP-IGMN when more data arrive the accuracy rate increases until it stabilizes after the presentation of about 5000 data points. The Diag-IGMN cannot fit the data correctly and does not obtain satisfying results, this is due to the fact that the restriction it assumes about the data is wrong for the simulated data. In Figure 3.5 is illustrated the projection of the training data on the two principal axes (of highest variance) with the correct classes assignments and the result of classification obtained by LP-IGMN. The results showed that LP-IGMN is very effective even with high-dimensional data.




(a)




(b)

Figure 3.2: Clustering results for (a) IGMN and (b) LP-IGMN. The color represents a group, the mean vectors are marked with a cross and the principal component found by LP-IGMN is illustrated by a black line.

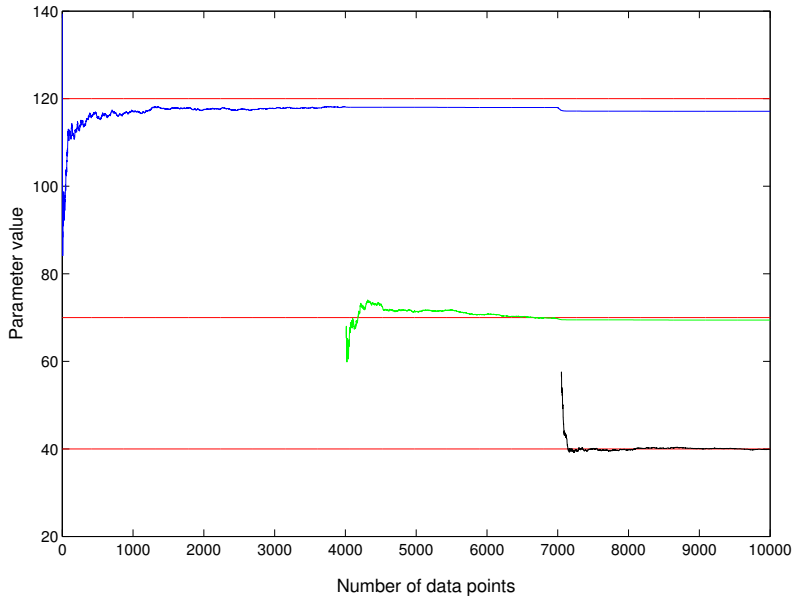



Figure 3.3: Evolution and convergence of the estimated parameters $a_k$ for a simulated dataset with 30-dimensional vectors. The horizontal red lines are the true values of the parameters.

### 3.2.3 Influence of the Dimensionality

In this experiment, our goal is to analyse the effect of the data dimensionality in the LP-IGMN. We generated 3 Gaussian distributions with $D$-dimensional feature vectors, for $D = 30, ..., 150$ with the parameters: $\{d_1, d_2, d_3\} = \{10, 10, 5\}$, $\{\pi_1, \pi_2, \pi_3\} = \{0.4, 0.3, 0.3\}$, $\{a_1, a_2, a_3\} = \{250, 75, 50\}$, $\{b_1, b_2, b_3\} = \{5, 5, 5\}$, close means and different covariance matrices. A set of $N = 1000$ data points are drawn from the Gaussian distributions above. The LP-IGMN was set with the same parameters as in the previous experiment. We then evaluate the performance of LP-IGMN running 10 times 5-fold cross validation (HAYKIN, 2008) with different datasets of size $N$ generated by the above Gaussian distributions. The 5-fold cross validation randomly divides the dataset into 5 subsets. The learning process is repeated 5 times using a different subset for testing (i.e., to compute the accuracy rate). This procedure is repeated 10 times and the final accuracy rate is calculated as the mean accuracy of the 10 replications. The classification performance for all datasets can be seen in Figure 3.6. LP-IGMN obtained very high performance with these data and therefore it is not influenced by the data dimensionality.

Note that the condition number of a matrix is the ratio of its largest and smallest eigenvalues. If a covariance matrix has a high condition number (ill-conditioned), for example, when one or more of the eigenvalues are zero, the distribution is singular and is confined to a subspace of lower dimensionality. In this case, an attempt to invert this covariance matrix fails due to numeric errors. This usually happens in IGMN learning process when the input vector has high dimensionality. However, note that this problem does not happen in LP-IGMN since it does not need to invert its covariance matrices, therefore it is robust for high-dimensional data. In order to analyse the condition number of the covariance matrices estimated by LP-IGMN during the learning process we performed an experiment using the first Gaussian distribution described above, which has condition number $c = 50$, since its largest

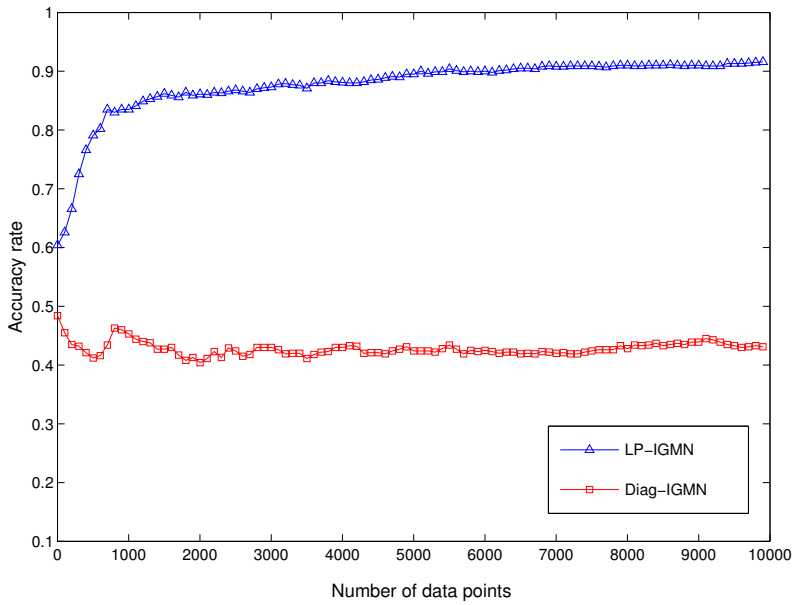
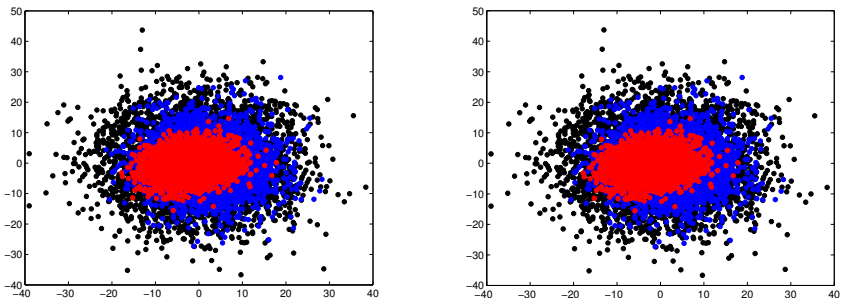



Figure 3.4: Evolution of the accuracy rate with the arrival of data points.



(a) (b)

Figure 3.5: (a) 30-dimensional data points projected on two first principal axes with the correct assignments of the data to the classes (each color represents a class) and (b) the data points assignments obtained by the LP-IGMN.

eigenvalue is 250 and its smallest is 5. First, consider $N = 1000$ data points in $\in \mathbb{R}^D$ drawn from this distribution. The condition number of the covariance matrix estimated by LP-IGMN is calculated as the ratio $\hat{a}_1/\hat{b}_1$. Figure 3.7 shows that the estimation of the condition number remains stable for different values of the data dimensionality $D$. We then fixed $D = 50$ and measured the condition number during the learning process to see the evolution of the estimated parameters. The result can be seen in Figure 3.8, which shows that when more data arrive the condition number of the covariance matrix approximates to the true value $c$. This is already expected due to the convergence of the parameters $a_k$ and $b_k$.

### 3.2.4 Configuring the parameter d

In the previous experiments the correct value of the parameter $d$ is known. In this section, we present some experiments to analyse how the LP-IGMN is affected by a wrong choice of the parameter $d$ and we show how to find the correct value of this parameter. We used the same 3 Gaussian distributions and the LP-IGMN parameters as in the previous experiment. For different datasets (each of size $N =$

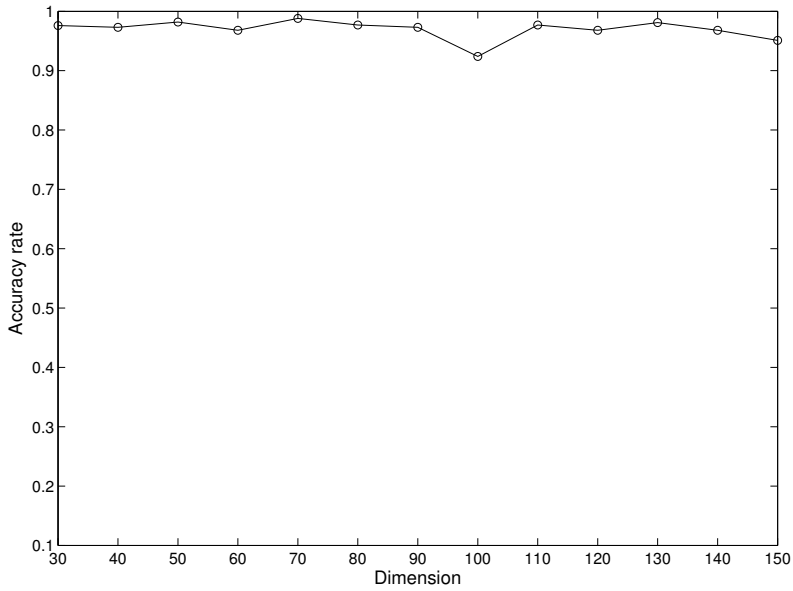



Figure 3.6: Influence of the data dimension on the accuracy rate.

1000) in $\mathbb{R}^D$, $D = 30, ..., 150$, drawn from the 3 Gaussian distributions, we trained the LP-IGMN with $d = 2, ..., 20$, and we evaluated the performance of the algorithm using 5-fold cross validation. The boxplot graph[1] of Figure 3.9 show the results. We know that the value $d = 10$ is the correct value for the generated data. It can be noticed that for $d < 10$ the performance is degraded since much information is discarded and for $d > 10$ the performance is high, but the variance is also high. This occurs because we are keeping both the dimensions that carry much information (of largest eigenvalues) and some dimensions that are noisy or which contain little information (of smallest eigenvalues). The correct value $d = 10$ yields the best result, achieving high performance for most models, keeping a small variation on the results (models with poor performance) and a mean accuracy rate of 94%.

In a real dataset, we can choose the value of the parameter $d$ using cross validation. To demonstrate this, we generated a dataset with a fixed data dimensionality $D = 130$ and evaluated the performance using 5-fold cross validation for different values of the parameter $d$. In Figure 3.10, we can observe that the performance decreases when we set wrong values of the parameter $d$, and for the correct value $d = 10$ there is a peak, where the algorithm achieves the highest performance. We will see in the next Section another way to choose the value of the parameter $d$, in which $d$ is chosen in order to keep a high value of the explained variance (e.g., 95% or 99% of the overall variance) and this leads to high performance.

### 3.2.5 High-dimensional Data Compression and Visualization

In this experiment, we highlight the ability of LP-IGMN in performing compression of high-dimensional data and its application for visualization of this type of data. Remember from (3.6) that the reconstruction of a data point in LP-IGMN is

[1] A boxplot is a statistical tool for graphically depicting groups of numerical data through their five-number summaries: the minimum value, lower quartile, median, upper quartile, maximum value and outliers (MASSART et al., 2005).

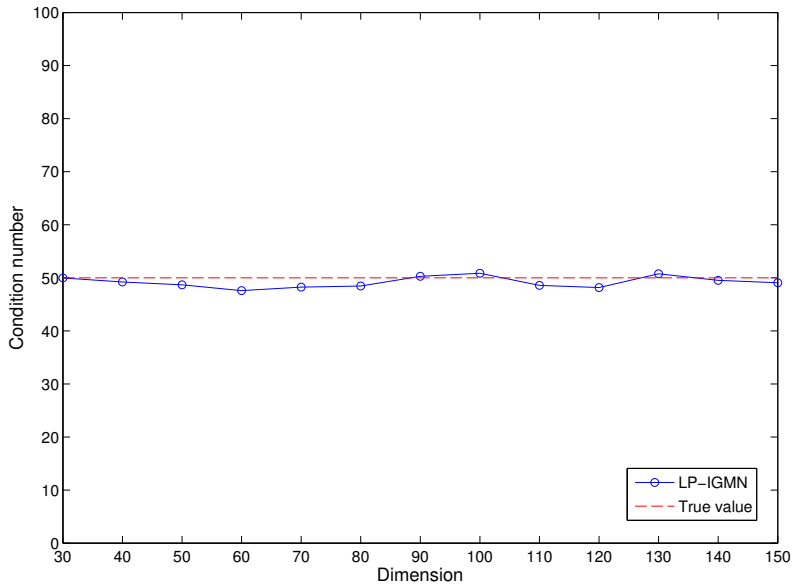



Figure 3.7: Influence of the data dimension on the condition number.

given by:

$$\tilde{\mathbf{x}}_n = \boldsymbol{\mu}_k + \sum_{i=1}^{d} (\mathbf{x}_n^T \mathbf{u}_{ki} - \boldsymbol{\mu}_k^T \mathbf{u}_{ki}) \mathbf{u}_{ki}. \tag{3.36}$$

This represents a compression of the dataset, because for each data point we have replaced the $D$-dimensional vector $\mathbf{x}_n$ with a $d$-dimensional vector having components $(\mathbf{x}_n^T \mathbf{u}_{ki} - \boldsymbol{\mu}_n^T \mathbf{u}_{ki})$. The smaller the value of $d$, the greater the degree of compression. To illustrate this process, we used the *Extended Yale Face Database B* (GEORGHIADES; BELHUMEUR; KRIEGMAN, 2001; LEE; HO; KRIEGMAN, 2005), which contains frontal face images of 38 individuals. We resized the cropped and normalized images to have size 32x32, therefore they are represented by 1024-dimensional vectors. The images were taken under varying illumination conditions. We used 64 images from the first person to train the LP-IGMN, which was set with the parameters: $\beta = 0.1$, $\delta = 0.35$, $v_{min} = 100$, $sp_{min} = 4$. The parameter $d$ is choosen in order to keep 99% of the overall variance of the data. Recall that the ratio between the sum of the eigenvalues from 1 up to $d$ and the sum of all the eigenvalues given us the explained variance $\sigma_{exp}$ of the data, i.e., $\sigma_{exp} = \frac{1}{Tr(\boldsymbol{\Sigma})} \sum_{i=1}^{d} \lambda_i$, where $\lambda_i$ and $Tr(\boldsymbol{\Sigma})$ are the eigenvalues and the trace of the covariance matrix estimated over the dataset, respectively. In this experiment, LP-IGMN was trained with the parameters described above and with $d = 20$. Since the mean vectors and eigenvectors estimated by LP-IGMN are vectors in the original $D$-dimensional space, we can represent them as images of the same size as the data points. Figure 3.11 shows these vectors as images. LP-IGMN found 8 subspaces in the data, whose mean vectors and the 20 eigenvectors associated to the largest eigenvalues that formed the first subspace are presented in Figure 3.11. We can observe that the estimated mean vector represents a frontal pose with different illumination conditions and the eigenvectors represent the illumination variation.

Another application to LP-IGMN is to high-dimensional data visualization. Here each data point is projected onto a two-dimensional principal subspace of a neuron component $k$, so that a data point $\mathbf{x}_n$ is plotted at Cartesian coordinates given

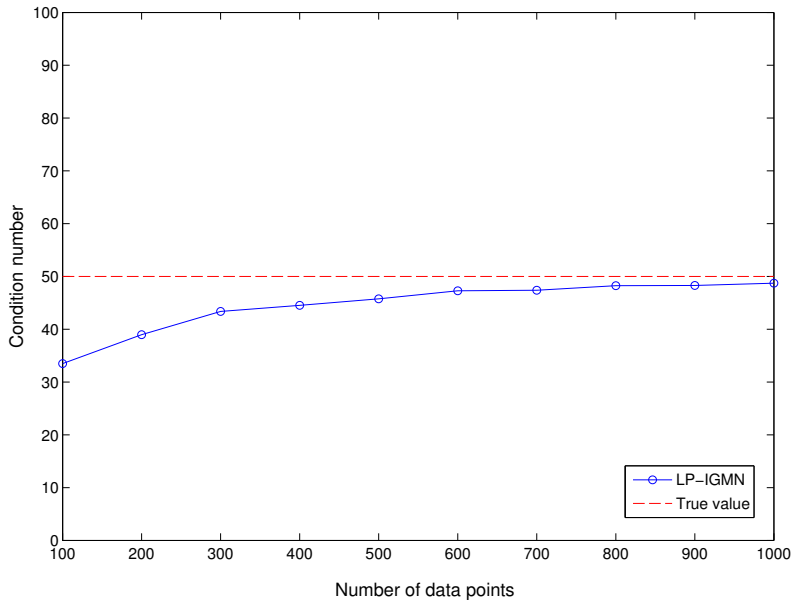



Figure 3.8: Condition number versus the number of observations.

by $\mathbf{x}_n^T \mathbf{u}_{k1}$ and $\mathbf{x}_n^T \mathbf{u}_{k2}$, where $\mathbf{u}_{k1}$ and $\mathbf{u}_{k2}$ are the eigenvectors corresponding to the largest and second largest eigenvalues. An example of such a plot, for the Iris data set (BACHE; LICHMAN, 2013), which contains data points in $\mathbb{R}^4$ space and 3 classes (Iris setosa, Iris versicolor and Iris virginica), is shown in Figure 3.12.

## 3.3 Summary

In this chapter we presented the algorithm LP-IGMN that was evaluated in several experiments that showed that LP-IGMN is robust to the data dimensionality, being able to model the density of low-dimensional and high-dimensional data. It avoids to invert an estimated covariance matrix and to compute eigenvectors associated to the smallest eigenvalues of an estimated covariance matrix, that are numerical unstable to estimate and hence it does not have numerical problems during the learning process. This numerical problem is the main drawback in the algorithm IGMN when dealing with high-dimensional data. LP-IGMN showed in the experiments that is able to find the correct subspaces where the data lies and can estimate incrementally the eigenvectors that form the subspaces, being efficient and suitable to perform online learning. Furthermore, LP-IGMN can be used as a visualization tool of high-dimensional data, since the data can be projected onto a two-dimensional principal subspace of highest variance.

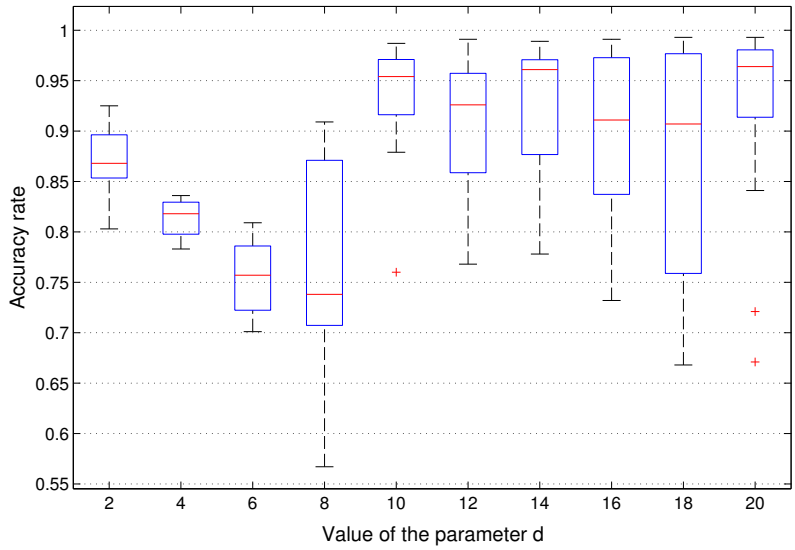



Figure 3.9: Impact of the parameter $d$ on the performance of the LP-IGMN with different input data dimensionality.

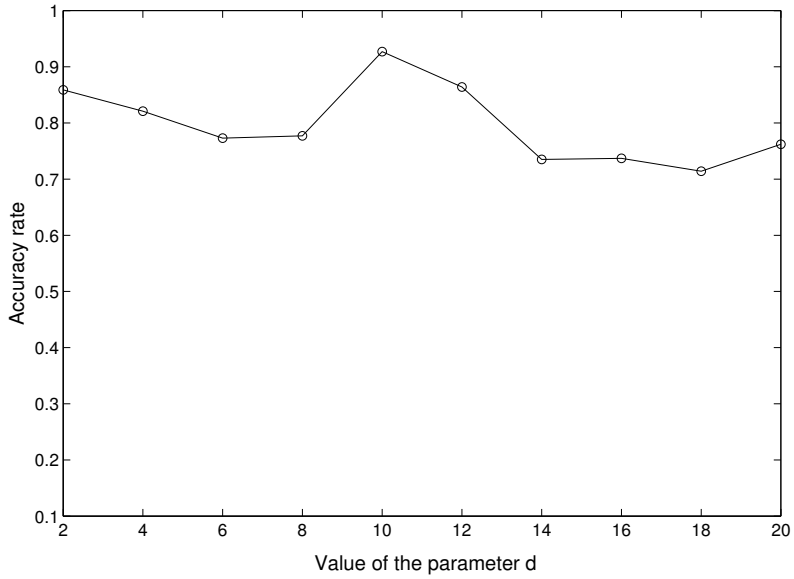
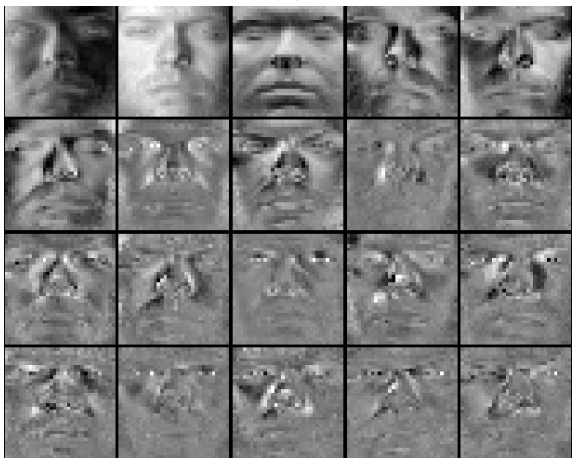


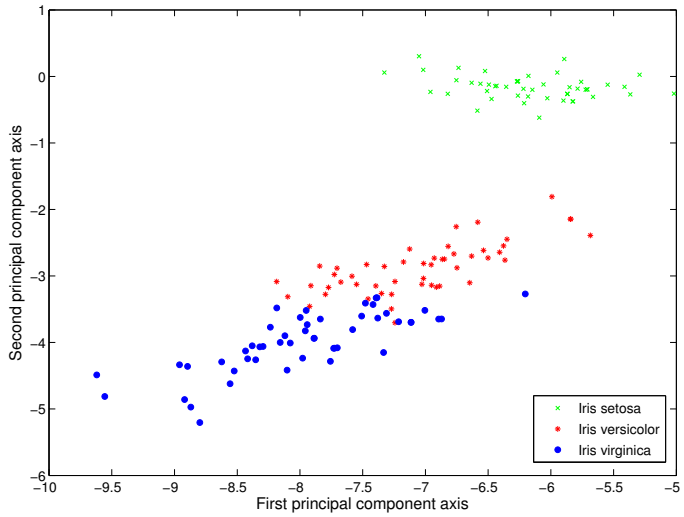

Figure 3.10: Choosing the parameter $d$ using cross validation.

(a)

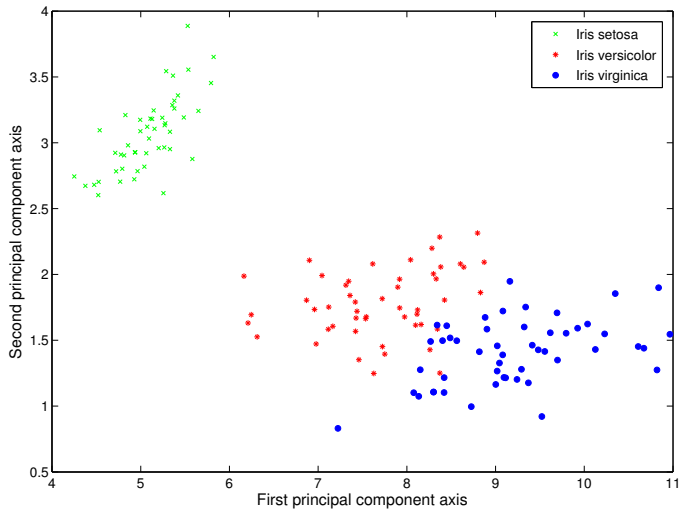


(b)

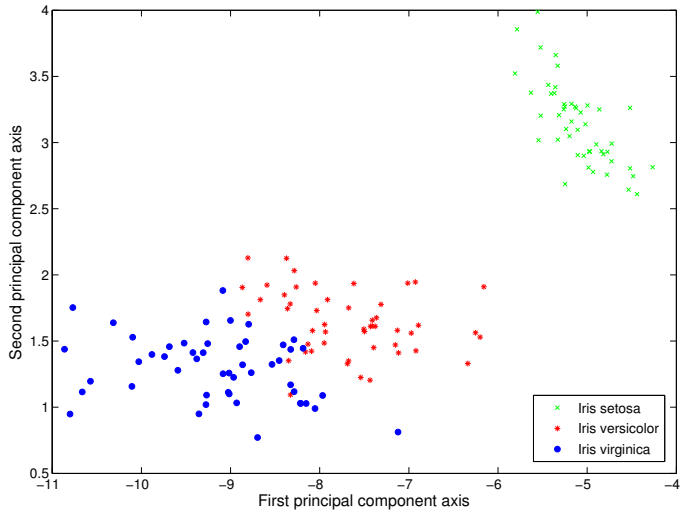
Figure 3.11: (a) the mean vectors estimated by LP-IGMN using the YALEFACES data from the first class and (b) the estimated eigenvectors of the first hidden neuron.




(a)




(b)




(c)

Figure 3.12: Projection of the Iris data points onto the two-dimensional principal subspace found by the (a) first hidden neuron, (b) second hidden neuron and (c) third hidden neuron of LP-IGMN.

# 4 APPLICATIONS FOR IMAGE RECOGNITION AND REPRESENTATION

Online learning of image representations is a challenging problem, since the system must be able to continuously update and increase its knowledge to perform useful tasks like face and object recognition. The raw input images can be represented by high-dimensional vectors, concatenating all the image pixels. These vectors can be used to train a classifier to learn a map $f : \mathbf{x} \longrightarrow y$ of the input image $\mathbf{x} \in \mathbb{R}^D$ to a class label $y$. Recent Machine Learning approaches use an intermediate feature descriptor $\phi(\mathbf{x}, \mathbf{\Theta})$ to extract high-level features of the image before training a classifier (CARVALHO; ENGEL, 2013). In order to show how the proposed method LP-IGMN can be used in image classification tasks, we restrict, without loss of generality, the input data to be the raw image pixels, but it is also possible to embed our algorithm in a more complex pipeline to learn intermediate and high-level features of the image, like the pipelines used by deep learning algorithms (BENGIO, 2009; HINTON; SALAKHUTDINOV, 2006). Usually the dimensionality $D$ of the input image is large (e.g., $D = 1024$) and many learning algorithms suffer with the so called *curse of dimensionality* (BELLMAN, 1961). For example, the IGMN fails to handle high-dimensional data, because its covariance matrices become singular and can not be inverted. This happens because many dimensions are noisy and the important information is found in a subspace of lower dimensionality. It has been shown in the previous chapter that the LP-IGMN is robust to the curse of dimensionality and it can model the density of these data correctly, achieving high performance in the experiments. In this chapter, we highlight some applications of the proposed LP-IGMN. In the experiments, the input data have high dimensionality, therefore the standard IGMN algorithm can not be applied and compared. Throughout all the experiments described in this chapter, the LP-IGMN parameters $\beta, \delta, v_{min}$ and $sp_{min}$ are found using grid search in order to achieve the best result on the test data. We also compare the LP-IGMN to other state-of-the-art models, and the results show that our method has competitive or better performance than many algorithms.

This chapter is organized as follows: Section 4.1 presents the experiments of the LP-IGMN for face recognition. Section 4.2 describes the experiments for handwritten digits recognition. In Section 4.3 is presented the evaluation of the proposed method in the task of object recognition. In Section 4.4 is detailed how to apply LP-IGMN for image denoising. Section 4.5 describes how our method can be used for image segmentation and Section 4.6 presents a summary.

## 4.1 Face Recognition

Face recognition is a challenging and important task. It can be a component in a security system or in a search tool of social nets, for example. The illumination conditions, occlusions, the different poses and the quantity of individuals (classes) arise several difficulties to learn a map between the image space and the labels (individuals). Furthermore, this map is highly nonlinear and simple optimizations algorithms can not achieve satisfactory results. Computationally intensive nonlinear optimization techniques must be used, and there is the risk of finding a suboptimal local minimum of the error function. As we have already noted, many natural sources of data carry many noisy dimensions and because of this the data live in a subspace of the higher dimensional observed data space. Capturing this property explicitly can lead to improved density modelling compared with more general methods. The face images have this property and we can use the LP-IGMN algorithm to capture it. In this experiment, we use the *Extended Yale Face Database B* (GEORGHIADES; BELHUMEUR; KRIEGMAN, 2001; LEE; HO; KRIEGMAN, 2005), which consists of 38 individuals and each with 64 frontal face images. The images were taken under different illumination conditions controlled in laboratory. The images are cropped and resized to have size 32x32, therefore the input data $\mathbf{x}$ is a 1024-dimensional vector. We also rescaled the pixels intensities to [0,1]. This normalization is necessary to avoid problems with the distance metric used by some algorithms that we use to compare with our method. A random set of 10 images each for 10 individuals is shown in Figure 4.1.

Figure 4.1: Random image samples for 10 individuals of the Extended Yale Face Database B. Each row represents an individual (class).

We follow the experiment setting in YANG; WANG; HUANG (2011), which uses a random set of 50 face images from each individual for training and the rest for test. Each training and testing step is repeated 10 times using different random sets of images and we measure the average accuracy rate and the standard deviation for the following algorithms: Naïve-Bayes (JOHN; LANGLEY, 1995), K-NN (AHA; KIBLER; ALBERT, 1991), Decision Trees (J48) (QUINLAN, 1993), Random For-

est (BREIMAN, 2001), Multilayer Perceptron (MLP) (RUMELHART; HINTON; WILLIAMS, 1986), Linear SVM (CORTES; VAPNIK, 1995) and the proposed LP-IGMN. We adjust parameters of the algorithms in order to maximize their performance on the test data. We train the K-NN with $k = 1$ (1 nearest neighbour); the Random Forest with 50 trees; the MLP with 100 hidden neurons and weight decay parameter (regularization parameter) $\lambda = 0.0001$ and we use 100 iterations of the Limited Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) (NOCEDAL, 1980) algorithm to optimize its parameters; we set the regularization parameter $C$ to 100 in the Linear SVM; and in LP-IGMN we set the parameters $\beta = 0.1$, $\delta = 0.35$, $v_{min} = 10$, $sp_{min} = 3$. We evaluate different values of the parameter $d$ in LP-IGMN in order to find the best configuration. Figure 4.2 shows the accuracy rate on the test data for various values of this parameter. It can be noted that the value of the parameter that yields the best performance is $d = 5$, for which the average accuracy rate is 97%.

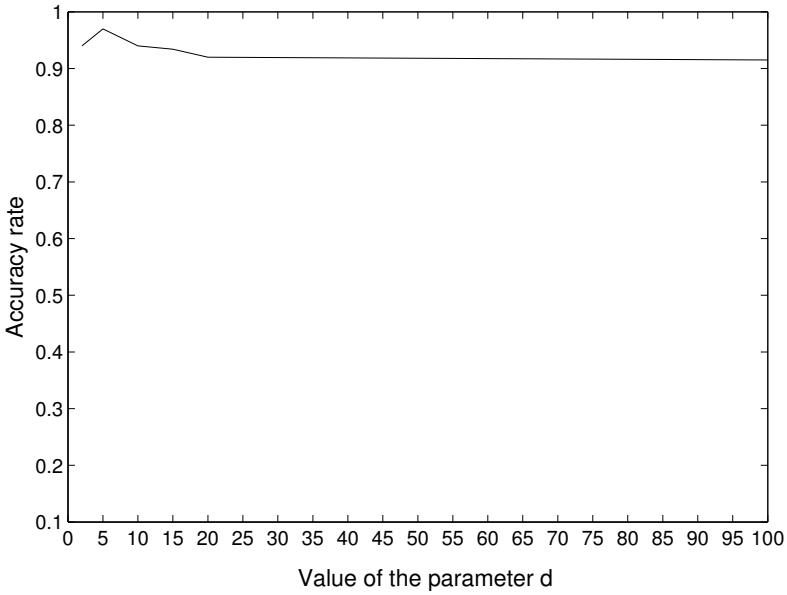
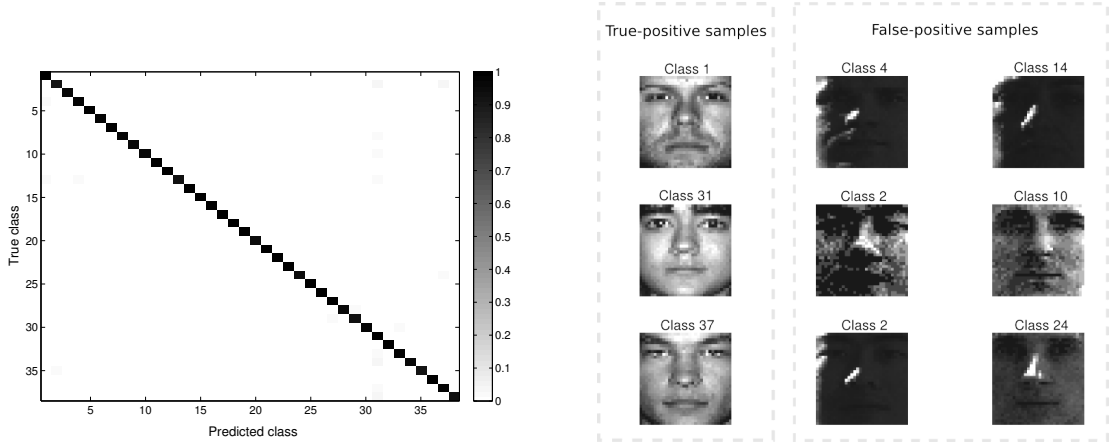



Figure 4.2: Accuracy rate on the Extended Yale Face B test data for different values of the parameter $d$. The value $d = 5$ yields the highest performance (97%).

Figure 4.3 illustrates the confusion table between the 38 individuals and the most confused samples (misclassified samples) for the individuals 1, 31 and 37, that occurs during the evaluation of the algorithm LP-IGMN. We can see that the confused images (false-positive samples) are very hard to classify because they have partial occlusions and low illumination.

In Table 4.1, we can see the LP-IGMN performance compared to other state-of-the-art algorithms. The algorithms Naïve-Bayes, K-NN and Decision Trees have bad results, since they suffer with the curse of dimensionality. The regularized models: Random Forest, MLP and Linear SVM achieve much better results. LP-IGMN outperforms these algorithms and is competitive with the Supervised SRC (YANG; WANG; HUANG, 2011). This method uses a computational complex algorithm (Sparse Coding) for supervised learning a set of $K = 20$ base vectors to represent each class in a high dimensional sparse space. In contrast, LP-IGMN learns just one set of bases, represented by its mean vectors in the hidden layer, and it adjusts the size of this set automatically in order to model the data density. The learned bases




(a) (b)

Figure 4.3: (a) Average confusion table for the LP-IGMN on the Extended Yale Face B dataset. The entry in the $i$th row and $j$th column is the percentage of images from class $i$ that were identified as class $j$, and (b) the true-positive samples from the classes 1, 31 and 37 and the most confused images (false-positive samples) for each class (row). The true class label is shown on the top of each image.

in LP-IGMN together with the eigenvectors estimated by the hidden neurons form the subspace where the data live. We can project the data in these subspaces to have an interesting insight into the classes. In Figure 4.4 is illustrated this process, where the test data is projected onto the principal subspace found by each neuron. We can see clearly the nonlinear nature of this dataset. Therefore, LP-IGMN can effectively learn the nonlinear map from the original high-dimensional space of the images to the class-specific space.

Table 4.1: Performance comparison of the LP-IGMN with several state-of-the-art methods on the Extended Yale Face B dataset.

| Algorithm | Accuracy (%) |
|---|---|
| Naïve-Bayes | 54.3 ± 2.4 |
| K-NN | 73.8 ± 1.5 |
| Decision Trees | 74.4 ± 2.6 |
| Random Forest | 96.1 ± 0.6 |
| MLP | 92.3 ± 0.9 |
| Linear SVM | 96 ± 0.6 |
| LP-IGMN (ours) | **97** ± 0.7 |
| Supervised SRC (YANG; WANG; HUANG, 2011) | **98.44** |

## 4.2 Handwritten Digits Recognition

The development of algorithms capable of automatically recognize handwritten digits is of great interest in the machine learning community and the industry. Efficient methods make possible the design of automate systems that deal with numbers such as postal code, banking account numbers and numbers on car plates. For example, post offices can use these methods to sort letters and the banks can use

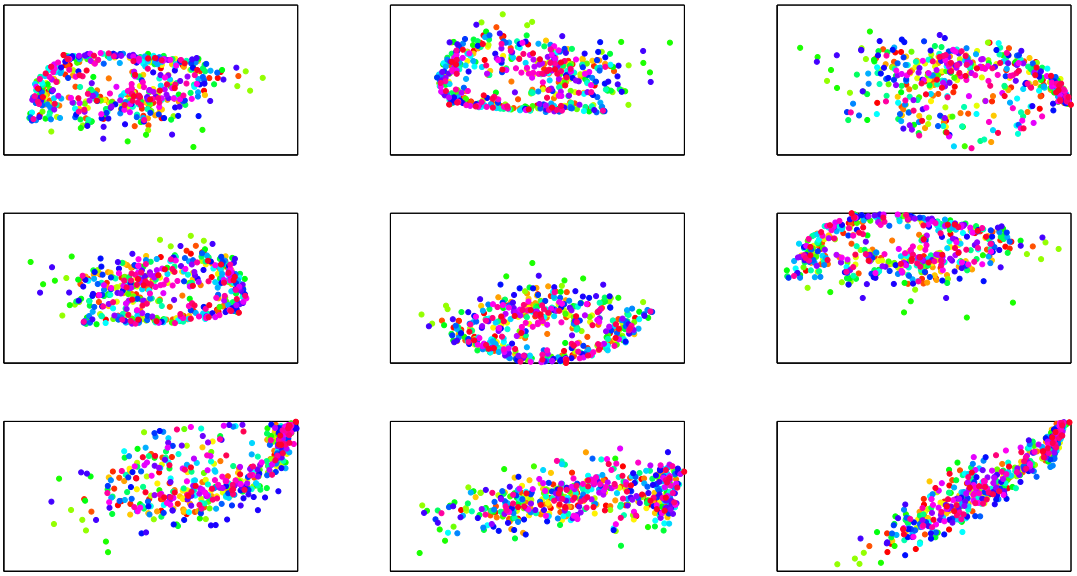


Figure 4.4: Projection of Extended Yale Face B test data on the 2 principal components of each neuron. Colors corresponds to different individuals (classes).

them to read personal checks. In order to show how the LP-IGMN can be applied in the task of handwritten digits recognition, we perform experiments on the USPS digits[1] dataset. This dataset contains 7291 training and 2007 test 16x16 gray-scale images. In Figure 4.5 is shown random samples of each class of the USPS Digits dataset.

Figure 4.5: Random samples of each class (row) of the USPS Digits data.

We randomly pick 500 training images per class and 200 test images per class. This procedure is repeated 10 times and we measure the accuracy rate and the standard deviation. We use the same parameters in LP-IGMN as in the previous Section. We found that the best value of the parameter $d$ for this dataset is $d = 10$, as can be viewed in Figure 4.6. Using this configuration in LP-IGMN, we achieve the average accuracy rate of 95.57%.

[1]http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html

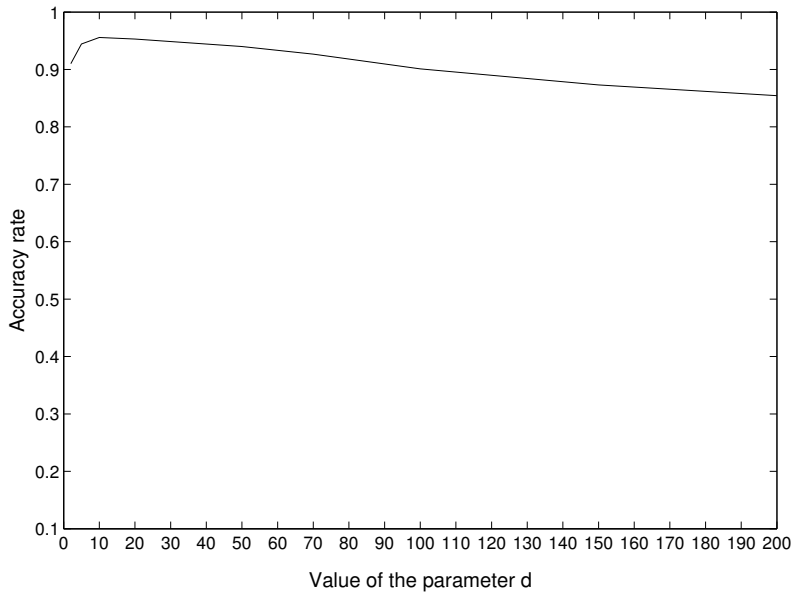



Figure 4.6: Accuracy rate on the USPS Digits test data for different values of the parameter $d$. The best value is $d = 10$, which yields the accuracy rate of 95.6%.

The average confusion table obtained from the evaluation of the LP-IGMN can be viewed in Figure 4.7. The diagonal shows the average accuracy rate per class. Most confusion occurs between the digits 3 and 5, and between the digits 7 and 9. However, the accuracy rate per class is still high (minimum of 92.7% in the class 3).

An interesting feature in LP-IGMN is that it allows the visualization of what has been learned by the model. In Figure 4.8 is shown the mean vectors, the first principal component associated to the largest eigenvalue and the second principal component associated to the second largest eigenvalue of each hidden unit in LP-IGMN after training on the USPS Digits data. The mean vectors represent blur versions of the digits and the principal components seem to capture the shape and angle variation of the digit. This meaningful representation offered by our algorithm can help researchers to find important structures and to have an insight of the concepts extracted from the raw data.

We compare the LP-IGMN with other state-of-the-art approaches, in particular, with the Linear SVM, MLP and Non-linear discriminative dictionary (NLDD) (SHRIVASTAVA et al., 2013). We use the following parameters: $C = 100$ (regularization parameter) in the Linear SVM; 50 hidden neurons, $\lambda = 0.0001$ (weight decay parameter) and 100 iterations of the L-BFGS in the MLP. The results can be seen in Table 4.2. LP-IGMN outperforms the Linear SVM and the MLP, and it remains competitive with the result reported in SHRIVASTAVA et al. (2013). The NLDD algorithm relies in the expensive kernel computation of a nonlinear mapping from the image space to a higher dimensional feature space. This mapping seems to yield better discriminative properties, but its computation cost is infeasible for online learning. On the other hand, LP-IGMN is suitable for online setting and to deal with large datasets, since its computational complexity is linear in the number of data points $N$.

| True class \ Predicted class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 97.35 | 0.15 | 0.65 | 0.10 | 0.15 | 0.75 | 0.40 | 0.10 | 0.25 | 0.10 |
| 1 | 0.00 | 98.50 | 0.20 | 0.00 | 0.55 | 0.10 | 0.15 | 0.00 | 0.50 | 0.00 |
| 2 | 0.40 | 0.20 | 96.00 | 0.85 | 0.70 | 0.20 | 0.15 | 0.65 | 0.75 | 0.10 |
| 3 | 0.50 | 0.10 | 1.05 | 92.70 | 0.05 | 3.55 | 0.10 | 0.35 | 1.50 | 0.10 |
| 4 | 0.40 | 0.40 | 0.55 | 0.00 | 94.00 | 0.45 | 0.65 | 0.95 | 0.60 | 2.00 |
| 5 | 0.50 | 0.10 | 0.30 | 2.05 | 0.20 | 93.75 | 0.80 | 0.25 | 1.35 | 0.70 |
| 6 | 0.70 | 0.15 | 0.20 | 0.10 | 0.70 | 0.65 | 97.05 | 0.05 | 0.40 | 0.00 |
| 7 | 0.05 | 0.15 | 0.30 | 0.35 | 0.90 | 0.15 | 0.00 | 94.45 | 0.75 | 2.90 |
| 8 | 0.70 | 0.25 | 0.75 | 0.80 | 0.50 | 0.45 | 0.20 | 0.95 | 95.05 | 0.35 |
| 9 | 0.05 | 0.10 | 0.25 | 0.15 | 1.10 | 0.10 | 0.00 | 0.90 | 0.45 | 96.90 |

Figure 4.7: Average confusion table for the LP-IGMN on the USPS Digits dataset. The diagonal values represent the percentage of correctly classified images for each class.

Figure 4.8: Top to bottom: the mean vectors; the first principal component (of largest eigenvalue); the second principal component (of second largest eigenvalue) of the hidden neurons in LP-IGMN after training on the USPS Digits data.

## 4.3 Visual Object Recognition

The task of identifying objects in images is of fundamental importance to vision systems and it has been investigated extensively. For example, intelligent robots need to identify objects to better understand the environment and to perform useful tasks like search for a specific object in complex terrains (SCHNEIDER et al., 2009; SAIDI et al., 2007). Several approaches that were developed to address the problem of visual object recognition rely in subspace analysis and use state-of-the-art methods such as PCA (JOLLIFFE, 2005), KPCA (SCHöLKOPF; SMOLA; MüLLER, 1998) and LDA (MCLACHLAN, 2004) in order to find a better image representation before training a classifier (LEE et al., 2005; LEONARDIS; BISCHOF, 2003). Following this line, we can apply the LP-IGMN algorithm and find the subspaces where each object data live. To demonstrate this, we performed several experiments with the well-known COIL-20 database (NENE; NAYAR; MURASE, 1996). This database consists of gray-scale images of 20 different objects rotated around one axis, where the 72 different views for each object are taken at pose intervals of 5 degrees. The objects of this dataset are presented in Figure 4.9. An example of the 72 views of the first object are displayed in Figure 4.10.

For our experiments we resized the original images to 16x16 pixels and therefore

Table 4.2: Performance comparison of the LP-IGMN with other state-of-the-art approaches on the USPS Digits test data.

| Algorithm | Accuracy (%) |
|---|---|
| Linear SVM | 88.5 ± 0.5 |
| MLP | 92.3 ± 0.8 |
| LP-IGMN (ours) | **95.6** ± 0.3 |
| NLDD (SHRIVASTAVA et al., 2013) | **97.5** |


Figure 4.9: Sample images of the COIL-20 database.

the input data is a 256-dimensional vector. Each input vector is normalized to have unit length. To evaluate our algorithm, we consider 36 random views of each object for training and remaining 36 views for testing. Therefore, we use 720 images for training and for testing. We set the LP-IGMN parameters as in the previous experiments. The performance of the proposed LP-IGMN with different values of the parameter $d$ can be seen in Table 4.3. These results represent the average accuracy rate for 10 random sets of training and testing images. Our best accuracy rate is 99.2%, which is achieved setting the parameter $d = 10$.

We also considered an online setting where only few samples are presented to the algorithm before testing. We assume that the presentation of 50 training images is one epoch, and after each epoch we classify the entire test data (720 images) to calculate the accuracy rate. The evolution of the LP-IGMN performance can be seen in Figure 4.11. The performance increases rapidly with the presentation of the training images. This shows that our method is suitable for online learning of object categories and can be used in this configuration by vision systems (e.g., by a robot's vision system).

We performed another experiment by varying the number of training views per object. The views are randomly selected and we use the best parameters determined in the previous analysis. The results can be seen in Table 4.4. We also compare these results with other state-of-the-art approaches, specifically with PCA, KPCA and

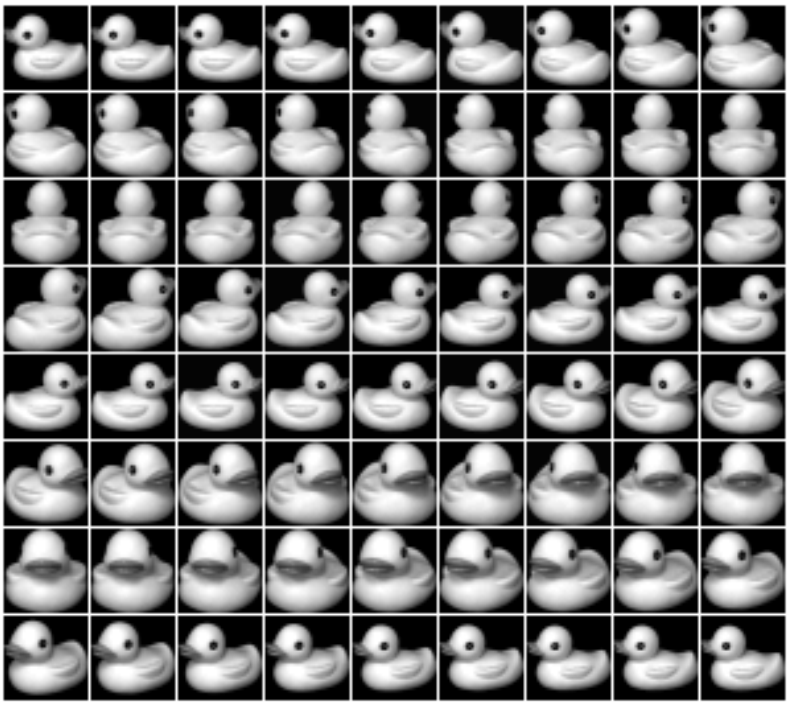


Figure 4.10: The 72 views of the first object in the COIL-20 database.

LDA. These methods are first used to find a lower dimensional representation of the input image and then a 1-nearest-neighbour classifier is used. All the methods are configured to find a 10-dimensional subspace of the COIL-20 images. The results are shown in Figure 4.12. LP-IGMN outperforms these algorithms under all conditions.

## 4.4 Image Denoising

The introduction of noise typically occurs during the image acquisition process and a denoising phase becomes essential to improve the image quality. Several methods have been developed to address this problem, from simple smoothing filters (GONZALEZ; WOODS, 2006) to the complex sparse representation (ELAD; AHARON, 2006). Many recent approaches instead of processing each pixel individually denoise blocks (or patches) of the image. These methods take into account the redundant information of small sub-images inside the image of interest to extract important features such as textures and borders, in order to use combinations of them to reconstruct a denoised image version. It has been shown that these patch-based methods are highly efficient for image denoising (DELEDALLE; SALMON; DALALYAN, 2011). Assuming that the noise $\varepsilon$ is white additive Gaussian with zero mean and standard deviation $\sigma$, i.e., $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, we can denote the observed corrupted image as $\boldsymbol{\gamma}_\varepsilon = \boldsymbol{\gamma} + \varepsilon$, where $\boldsymbol{\gamma}$ is the unobserved true image vector. The goal of denoising is to obtain an estimation $\tilde{\boldsymbol{\gamma}}$ from the observation $\boldsymbol{\gamma}_\varepsilon$ such that $\tilde{\boldsymbol{\gamma}}$ is close to $\boldsymbol{\gamma}$.

Considering the patch-based approach, we can treat the observed noise image $\boldsymbol{\gamma}_\varepsilon$ as a collection of non-overlapping blocks of size $D = w.w$, i.e., as a set $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n, ..., \mathbf{x}_N\} \in \mathbb{R}^{D \times N}$, where $w$ is an user defined parameter, which is usually chosen by cross validation. One simple approach to estimate $\tilde{\boldsymbol{\gamma}}$ is to use PCA over the set of patches, drop the noise components (i.e., the noise dimensions) and finally

Table 4.3: Classification accuracies for different values of the parameter $d$ in LP-IGMN.

| Value of the parameter d | Accuracy (%) |
|---|---|
| 2 | 96.7 |
| 5 | 98.6 |
| 10 | 99.2 |
| 20 | 98.8 |
| 50 | 98.5 |
| 70 | 98.2 |
| 100 | 97.0 |
| 150 | 95.3 |
| 200 | 93.6 |

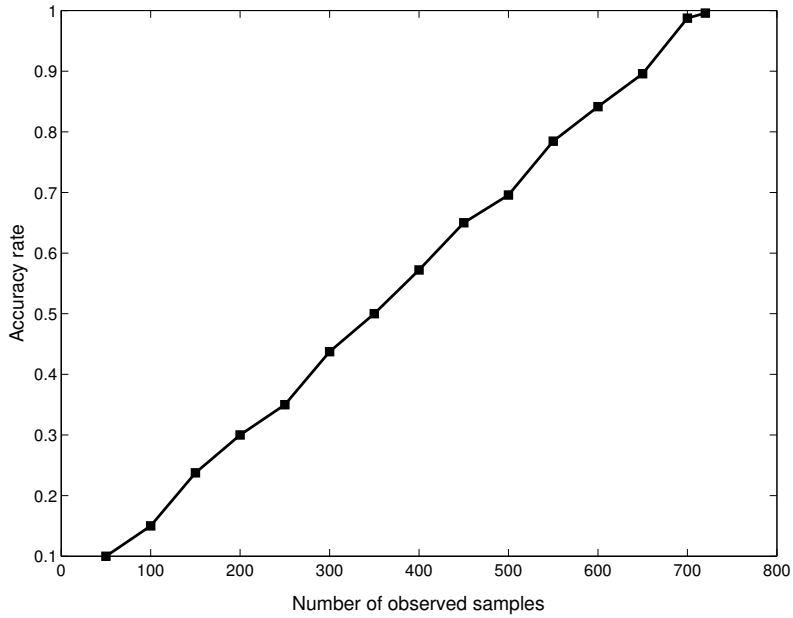



Figure 4.11: Performance evolution in the COIL-20 database.

reproject these data onto the original image space. We use a similar approach to show how LP-IGMN can be applied to perform denoising in images. First, we extract $M$ random patches from the corrupted image $\boldsymbol{\gamma}_\varepsilon$ and we use them to unsupervised training the LP-IGMN. In our experiments, we use $M = 50000$ random patches. Next the learned model is used to calculate the data point approximation $\tilde{\mathbf{x}}_n$ of each non-overlapping patch $\mathbf{x}_n$. In other words, our model reconstructs the patches to create an estimation $\tilde{\boldsymbol{\gamma}}$ of the free-noise image $\boldsymbol{\gamma}$.

Since in LP-IGMN a different data point approximation is given by each hidden neuron because the input vector can be projected locally onto the principal subspace found by a hidden unit, we have to consider a global approximation for this data in the order to reconstruct an image patch. We calculate the global approximation using equation (3.7).

In Figure 4.13 is displayed a set of random patches corrupted with a noise level $\sigma = 20$ and the reconstruction of these patches obtained by the LP-IGMN.

Table 4.4: Classification accuracies for different number of training views per object.

| Number of views used for training | Accuracy (%) |
|---|---|
| 2 | 73.0 |
| 4 | 81.0 |
| 8 | 90.4 |
| 18 | 96.5 |
| 36 | 99.2 |

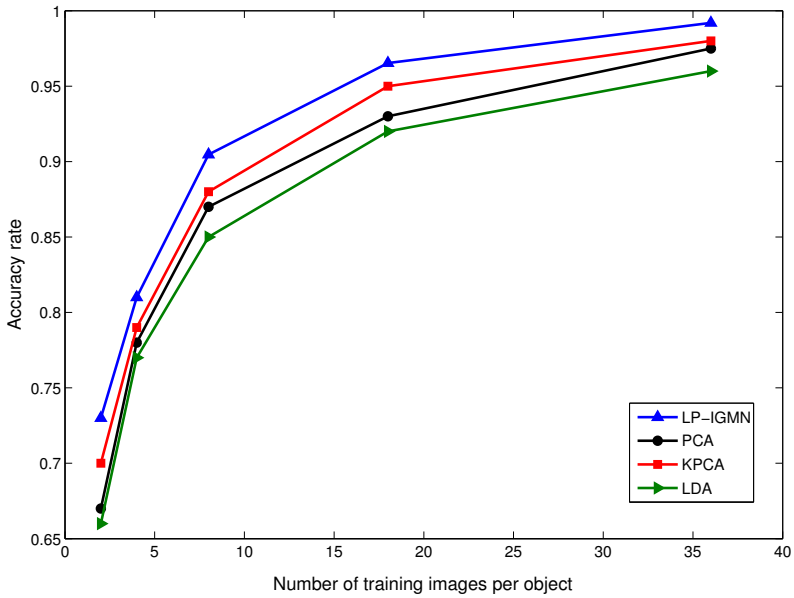


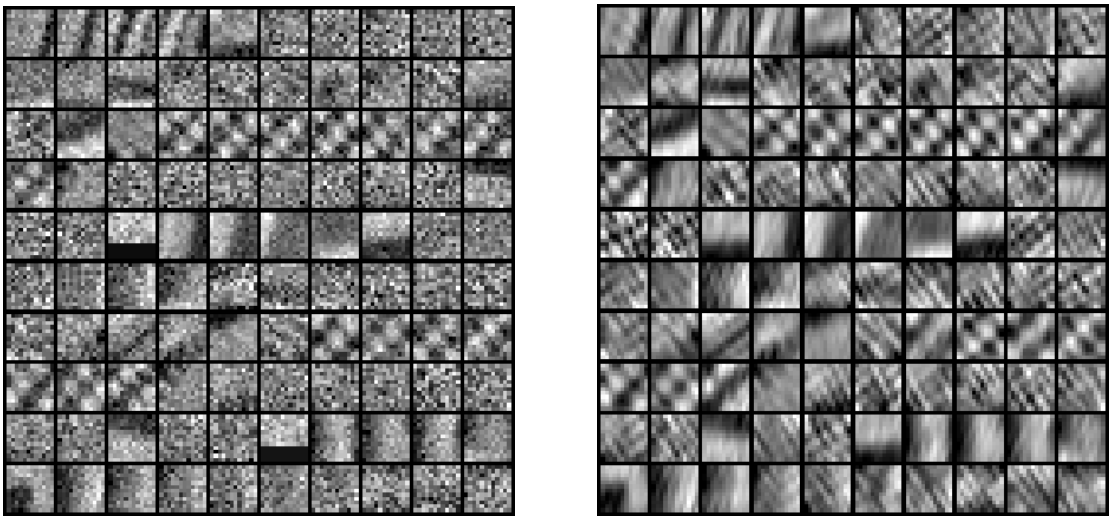

Figure 4.12: Recognition accuracies of LP-IGMN versus PCA, KPCA and LDA on the COIL-20 database, with a $d = 10$ dimensional principal subspace. The number of training images (views) per object were randomly selected from the 72 images of each object.

In our experiments, we set $w = 12$ as a compromise between aggregating enough information and the computational cost. Training with small windows (e.g., $w = 2$, $w = 4$) is fast, but they may not carry enough information about the structures such as textures, resulting in bad results. Large windows (e.g., $w = 10$, $w = 12$) increase the computational cost, but the results are considerably better in our experiments. The parameters in LP-IGMN are configured as following: $\beta = 0.1$, $\delta = 0.35$, $v_{min} = 5$, $sp_{min} = 3$.

To evaluate our approach, we use the Peak Signal to Noise Ratio (PSNR) as an accuracy measure, which is defined as:

$$PSRN(\tilde{\gamma}, \gamma) = 10 \log_{10} \frac{255^2}{||\tilde{\gamma} - \gamma||^2}. \tag{4.1}$$

We also compare our method with PCA. The algorithms are configured to keep the same number of components. In Figure 4.14 is shown the image *Barbara* damaged with noise level $\sigma = 10$ (first row) and $\sigma = 20$ (second row), and the corresponding denoised images obtained by PCA and LP-IGMN. To evaluate these methods quantitatively, we measure the performance of them for denoising the gray-scale test



(a) (b)

Figure 4.13: (a) A set of 100 random patches damaged with noise level $\sigma = 20$ and (b) the reconstructed (denoised) patches obtained by LP-IGMN.

images (of size 512x512) displayed in Figure 4.15.

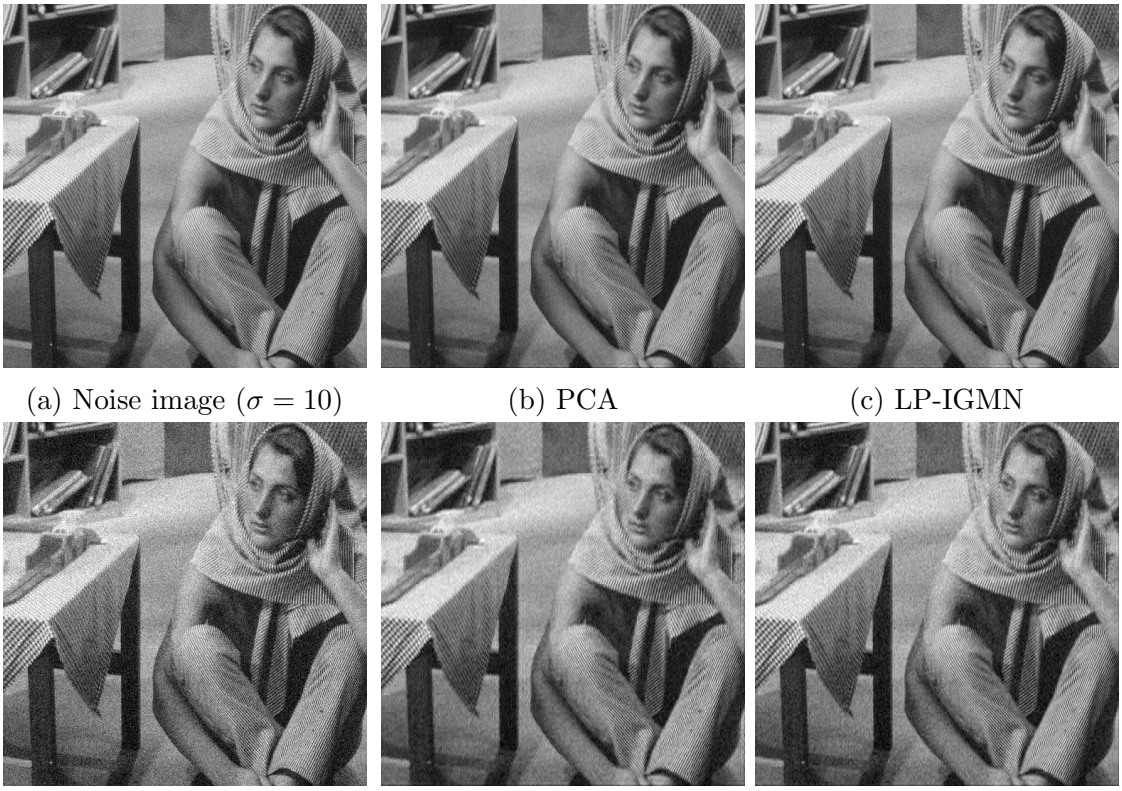
The results can be seen in Table 4.5. It can be noted that the LP-IGMN achieves good results and has performance similar or better than PCA. Therefore, our proposed method is suitable for denoising images.

Table 4.5: Performance comparison of the LP-IGMN with PCA for different noise levels.

| | PCA | LP-IGMN | Number of components |
|---|---|---|---|
| | $\sigma = 10$ | | |
| Barbara (PSNR=28.12) | 30.10 | 30.40 | 50 |
| Boat (PSNR=28.13) | 30.40 | 30.43 | 50 |
| Lena (PSNR=28.13) | 31.20 | 31.38 | 50 |
| Peppers (PSNR=28.13) | 33.21 | 33.26 | 20 |
| | $\sigma = 20$ | | |
| Barbara (PSNR=22.10) | 25.35 | 26.38 | 30 |
| Boat (PSNR=22.13) | 25.79 | 26.69 | 20 |
| Lena (PSNR=22.10) | 28.10 | 28.15 | 20 |
| Peppers (PSNR=22.11) | 29.45 | 30.16 | 10 |
| | $\sigma = 30$ | | |
| Barbara (PSNR=18.58) | 23.52 | 23.52 | 20 |
| Boat (PSNR=18.61) | 24.91 | 24.91 | 10 |
| Lena (PSNR=18.57) | 26.50 | 26.50 | 10 |
| Peppers (PSNR=18.59) | 28.20 | 28.20 | 10 |

## 4.5 Color Image Segmentation

Color image segmentation is an important and useful task that has many applications. The segmentation allows us to identity regions and objects of interest in



(a) Noise image ($\sigma = 10$) (b) PCA (c) LP-IGMN

(d) Noise image ($\sigma = 20$) (e) PCA (f) LP-IGMN

Figure 4.14: First row: (a) a noise image (PSNR=28.12) damaged with noise level $\sigma = 10$ and the corresponding denoised image obtained (b) by PCA (PSNR=30.10) and (c) by LP-IGMN (PSNR=30.40). Second row: (d) a noise image (PSNR=22.10) damaged with noise level $\sigma = 20$ and the corresponding denoised image obtained (e) by PCA (PSNR=25.35) and (f) by LP-IGMN (PSNR=26.38).

complex and confused scenes for posterior analysis. If an image only contains homogeneous color regions, unsupervised learning in color space can be applied efficiently. To demonstrate how the LP-IGMN can be applied to the problem of color image segmentation, we use the *Breast Cancer* dataset from UCSB (GELASCA et al., 2008), which contains RGB images stained with hematoxyling and eosin (H&E) used in breast cancer cell detection. These images are stained since most cells are essentially transparent, with little or no intrinsic pigment. Certain special stains, which bind selectively to particular components, are being used to identify biological structures such as cells. In those images, the challenging problem is cell segmentation for subsequent classification in benign and malignant cells. In this experiment, we only consider the problem of cell segmentation. In Figure 4.16 is shown an image of breast tissue, which contains malignant cells. In this image, we can see that there are three dominant colors (without considering brightness): white, pink and blue, and they are easy to be distinguished of each other by us. Because of this, we considered to use the $L^*a^*b^*$ color space instead of the RGB color space, since the $L^*a^*b^*$ color space was designed to approximates to the human perception. The $L^*a^*b^*$ space consists of a luminosity layer $L^*$ and two chromatics layers $a^*$ and $b^*$. The layer $a^*$ indicates where color falls along the red-green axis, and the layer $b^*$ indicates where the color falls along blue-yellow axis. Since all the color information

(a) (b) (c) (d)

Figure 4.15: The test images (a) *Barbara*, (b) *Boat*, (c) *Lena* and (d) *Peppers*.

are present in the a*b* space, we can measure distances between pixels using its values for a* and b*.

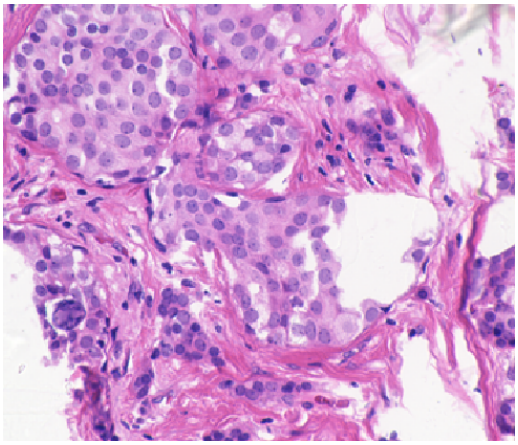


Figure 4.16: A breast tissue image containing malignant cells.

In order to identify the groups of colors in an image, we train the LP-IGMN considering the image pixels as input vectors in $\mathbb{R}^2$ containing the values of a* and b*. In this experiment, we used the tissue image presented in Figure 4.16 to segment the cells. The parameters of LP-IGMN was set as: $\beta = 0.0001$, $\delta = 0.1$, $v_{min} = 150$, $sp_{min} = 7$ and $d = 1$. LP-IGMN created 3 hidden neurons to represent the 3 groups of colors as we expected.

The image pixels are classified as belonging to a group $k$ using the maximum a posteriori rule (MAP). In Figure 4.17 is shown a segmentation mask obtained by the classification of the image pixels. This mask contains the labels (group index), represented with different colors (black, white and gray), for each pixel in an image. This mask can be used to segment the objects from the original RGB image. To perform this, we applied the mask over the tissue image in order to select only the pixels labeled as belonging to a group $k$. The results of this step are illustrated in Figure 4.18. Each image contains the objects that belong to a group $k$. Note that one of the groups in Figure 4.17 has light and dark blue objects. The cell nuclei is dark blue and therefore we need to separate it from the light blue objects. In order to perform this, we applied the Otsu's method (OTSU, 1979) with the blue pixels brightness values, which are obtained by the L* layer in the L*a*b* space, in order to find a threshold to separate the dark and light blue pixels. The result of the cell segmentation is shown in Figure 4.19.

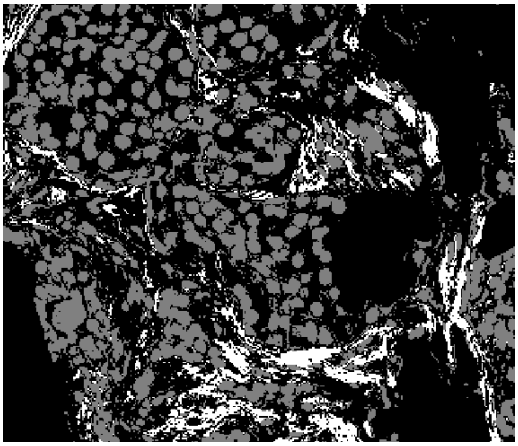
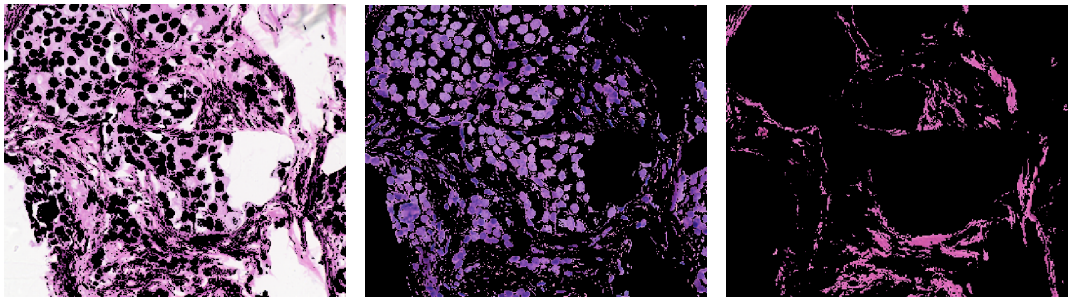


Figure 4.17: Breast tissue image segmentation mask containing the pixels labels.



(a) (b) (c)

Figure 4.18: Segmentated objects of the breast tissue image. (a) Segmented objects by the first neuron (first group), (b) segmented objects by the second neuron (second group) and (c) segmented objects by the third neuron (third group).

## 4.6 Summary

In this chapter we presented a wide variety of applications for our proposed LP-IGMN. We conducted several analysis with our algorithm in classification tasks and visualization of high-dimensional input data, and the results have demonstrated that LP-IGMN achieves high performance and outperforms several state-of-the-art methods. An explanation for the good results is that most real data carry useless information in many variables (or dimensions) and is necessary to find the intrinsic dimensionality or the subspace where the important information is located in order to achieve good results. From our experiments, it can also be noted that the configuration of the parameters in LP-IGMN is straightforward. The value of the most important parameter $d$, which defines the dimensionality of the principal subspace, can be found using cross validation, and a large range of values can yield satisfactory results. The other parameters, inherited from IGMN, have remained almost the same among the experiments, this highlights that they are not critical to achieve good results. Another important feature emphasized in our experiments is the capability of our algorithm to offer an interesting insight of what has been learned. The mean vectors and the eigenvectors in the hidden layer can be visualized as meaningful representations of the concepts extracted from the raw data when the input patterns are images. Furthermore, high-dimensional data can be projected locally onto a 2-dimensional space spanned by the first 2 principal components (eigenvec-

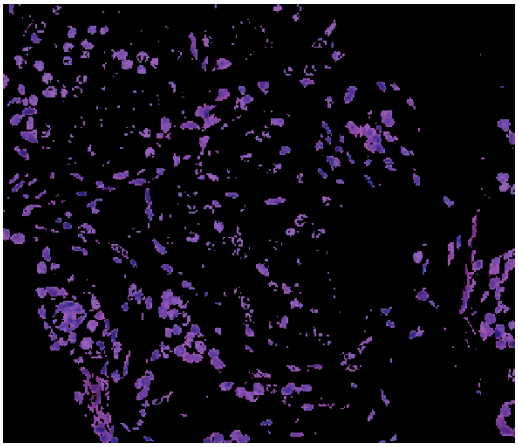


Figure 4.19: Cell segmentation of the breast tissue image.

tors) of a hidden neuron to give us an interesting insight about the data. We have also shown that the LP-IGMN global projection of the data point, which considers the contributions of each hidden unit to reconstruct an input pattern, can be used for image denoising tasks.

# 5 CONCLUSIONS

This monograph has presented LP-IGMN, a new artificial neural network able to model high-dimensional data streams efficiently, which is the main contribution of this dissertation. To evaluate our proposed model, we conducted several experiments using simulated and real datasets, and these experiments have demonstrated that: (i) LP-IGMN can find the specific subspace around which each neuron is located, representing the intrinsic dimensionality of the data; (ii) its parametrization is robust with respect to the ill-conditioning or the singularity of empirical covariance matrices, overcoming the curse of dimensionality problem that happens with IGMN; (iii) the noise variance which is orthogonal to the principal subspace is modelled by a single parameter in each hidden unit and this assumption provided very satisfying results for many types of data; (iv) the concepts learned from raw images are meaningful and can visualized plotting the LP-IGMN mean vectors and the eigenvectors as images in the original space; and (v) the LP-IGMN performance is competitive, and usually superior, with other state-of-the-art algorithms.

LP-IGMN was also tested in practical applications such as: (i) face recognition; (ii) handwritten digits recognition; (iii) object recognition; (iv) image denoising; and (v) image segmentation, and these experiments have shown that LP-IGMN can be used successfully in applications that require life-long learning in high-dimensional spaces. Specifically, the experiments have demonstrated that LP-IGMN has the following advantages over other standard approaches:

**LP-IGMN is suitable for life-long learning.** LP-IGMN automatically adjusts its topology in order to model the data density and can continuously acquire new knowledge considering the stability of the already learned knowledge. In addition, it does not have a separate phase for learning and recalling.

**LP-IGMN learning algorithm is fast and low memory consuming.** LP-IGMN learns using a single scan over the training data and each data point can be immediately used and discarded, therefore, there is no need to store the whole dataset in memory.

**The configuration of the parameters in LP-IGMN is straightforward.** A large range of values of the parameter $d$, which defines the dimensionality of the principal subspace, yields satisfactory results. The other parameters, inherited from IGMN, have remained almost the same among the experiments, this highlights that they are not critical to achieve good results.

**LP-IGMN can be used as a visualization tool of high-dimensional data.** The eigenvectors estimated by each hidden unit span a subspace where the data live

and the important information is located. We can project the whole dataset in these subspaces to have an interesting insight into the classes and geometrical structures of the data.

**LP-IGMN has suitable computational performance for online learning.** In online learning, the data come in flows, i.e., the data points arrive over the time and the number of data points $N \to \infty$. Unlike other complex algorithms, LP-IGMN has computational complexity linear in $N$ and, therefore, it can be applied efficiently in online settings.

**LP-IGMN can learn even in presence of redundant and/or irrelevant input dimensions.** Since LP-IGMN is a regularized model that can model the noise information and the dimensions with small variance using a single parameter in each hidden neuron, the learning algorithm is robust to different sources of data, including those that are very noise such as vision sensors in robots.

There are some directions in which this work can be continued:

- Verifying the benefits of allowing each hidden neuron in LP-IGMN finds a subspace with different intrinsic dimensionality $d_k$. If possible, let the model automatically find these values;
- Investigating ways to create and merge components in order to apply LP-IGMN in non-stationary environments, where there is no local context in the input data;
- Learning hierarchies of features before classifying. It is possible to extract more powerful features from the raw data stacking several hidden layers as it is done in the successful deep learning algorithms (BENGIO, 2009; HINTON; SALAKHUTDINOV, 2006);
- Verifying the possibility of learning independent components instead of linear components;
- Extending LP-IGMN to handle temporal high-dimensional sequences.

# REFERENCES


AHA, D.; KIBLER, D.; ALBERT, M. Instance-based learning algorithms. **Machine learning**, [S.l.], v.6, n.1, p.37–66, 1991.

BACHE, K.; LICHMAN, M. **UCI Machine Learning Repository**. 2013.

BELLMAN, R. E. **Adaptive Control Processes**: a guided tour. Princeton, New Jersey, U.S.A.: Princeton University Press, 1961.

BENGIO, Y. Learning Deep Architectures for AI. **Found. Trends Mach. Learn.**, Hanover, MA, USA, v.2, n.1, p.1–127, Jan. 2009.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. [S.l.]: Springer-Verlag New York, Inc., 2006.

BOUVEYRON, C.; GIRARD, S.; SCHMID, C. High-Dimensional Data Clustering. **Computational Statistics and Data Analysis**, [S.l.], v.52, n.1, p.502–519, 2007.

BREIMAN, L. Random forests. **Machine learning**, [S.l.], v.54, n.1, p.5–32, 2001.

CARVALHO, E. F.; ENGEL, P. M. Convolutional Sparse Feature Descriptor for Object Recognition in CIFAR-10. In: BRAZILIAN CONFERENCE ON INTELLIGENT SYSTEMS (BRACIS). **Proceedings...** [S.l.: s.n.], 2013. p.131–135.

CHRISTIAN, S. F.; LEBIERE, C. The Cascade-correlation Learning Architecture. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Proceedings...** Morgan Kaufmann, 1990. p.524–532.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine Learning**, [S.l.], v.20, n.3, p.273–297, 1995.

DELEDALLE, C.-A.; SALMON, J.; DALALYAN, A. S. Image denoising with patch based PCA: local versus global. In: BRITISH MACHINE VISION CONFERENCE (BMVC). **Proceedings...** [S.l.: s.n.], 2011. p.1–10.

DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern Classification**. 2.ed. New York, NY, USA: John Wiley, 2001.

ELAD, M.; AHARON, M. Image denoising via sparse and redundant representations over learned dictionaries. **IEEE Transaction on Image Processing**, [S.l.], v.15, n.12, p.3736–3745, 2006.

ENGEL, P. M.; HEINEN, M. R. Concept Formation Using Incremental Gaussian Mixture Models. **Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications**, [S.l.], p.128–135, 2010.

ENGEL, P. M.; HEINEN, M. R. Incremental Learning of Multivariate Gaussian Mixture Models. In: BRAZILIAN SYMPOSIUM ON AI (SBIA): ADVANCES IN ARTIFICIAL INTELLIGENCE, 20., São Bernardo do Campo, SP, Brazil. **Proceedings...** Springer-Verlag, 2010. p.82–91. (LNCS, v.6404).

FUKUNAGA, K. **Introduction to Statistical Pattern Recognition**. 2.ed. New York, NY, USA: Academic Press, 1990.

GELASCA, E. D. et al. Evaluation and Benchmark for Biological Image Segmentation. In: IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING. **Proceedings...** [S.l.: s.n.], 2008. p.1816–1819.

GEORGHIADES, A.; BELHUMEUR, P.; KRIEGMAN, D. From few to many: illumination cone models for face recognition under variable lighting and pose. **IEEE PAMI**, [S.l.], v.23, n.6, p.643–660, 2001.

GHAHRAMANI, Z.; JORDAN, M. I. Supervised Learning from Incomplete Data Via an EM Approach. **Advances in Neural Information Processing Systems**, San Francisco, CA, USA, v.6, 1994.

GONZALEZ, R.; WOODS, R. **Digital Image Processing**. 3.ed. [S.l.]: Prentice-Hall, Inc., 2006.

HALL, P.; MARSHALL, D.; MARTIN, R. Incremental Eigenanalysis for Classification. **British Machine Vision Conference (BMVC)**, [S.l.], v.1, p.286–295, 1998.

HAYKIN, S. **Neural Networks and Learning Machines**. 3.ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2008.

HEINEN, M. R. **A Connectionist Approach for Incremental Function Approximation and On-line Tasks**. 2011. 167p. Ph.D. Thesis — Informatics Institute – Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the Dimensionality of Data with Neural Networks. **Science**, [S.l.], v.313, n.5786, p.504–507, 2006.

HORNIK, K. Approximation Capabilities of Multilayer Feedforward Networks. **Neural Networks**, New York, NY, USA, v.4, 1991.

JOHN, G.; LANGLEY, P. Estimating continuous distributions in bayesian classifiers. In: CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE (UAI), 7., San Francisco, CA, USA. **Proceedings...** Morgan Kaufmann Publishers Inc, 1995. p.338–345.

JOHNSON, R. A.; WICHERN, D. W. **Applied multivariate statistical analysis**. 6 ed.ed. [S.l.]: Pearson, 2007.

JOLLIFFE, I. **Principal Component Analysis**. [S.l.]: Wiley Online Library, 2005.

KIRSTEIN, S. **Interactive and life-long learning for identification and categorization tasks**. 2010. Tese (Doutorado em Ciência da Computação) — Ilmenau University of Technology.

LEE, J. et al. Visual object recognition using probabilistic kernel subspace similarity. **Pattern Recognition**, [S.l.], v.38, n.7, p.997–1008, 2005.

LEE, K.; HO, J.; KRIEGMAN, D. Acquiring Linear Subspaces for Face Recognition under Variable Lighting. **IEEE PAMI**, [S.l.], v.27, n.5, p.684–698, 2005.

LEONARDIS, A.; BISCHOF, H. Kernel and subspace methods for computer vision. **Pattern Recognition**, [S.l.], v.36, n.9, p.1925—1927, 2003.

MASSART, D. L. et al. Practical Data Handling: visual presentation of data by means of box plots. **LC–GC Europe**, Santa Monica, CA, v.18, n.4, p.215–218, Apr. 2005.

MAXWELL, K. H.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural networks**, [S.l.], v.2, n.5, p.359–366, 1989.

MCLACHLAN, G. J. **Discriminant Analysis and Statistical Pattern Recognition**. [S.l.]: Wiley-Interscience, 2004.

NENE, S. A.; NAYAR, S. K.; MURASE, H. **Columbia Object Image Library (COIL-20)**. [S.l.]: Columbia University, 1996. (CUCS-005-96).

NOCEDAL, J. Updating quasi-Newton matrices with limited memory. **Mathematics of computation**, [S.l.], v.35, n.151, p.773–782, Jul 1980.

OTSU, N. A Threshold Selection Method from Gray-Level Histograms. **IEEE Transactions on Systems, Man and Cybernetics**, [S.l.], v.9, n.1, p.62–66, 1979.

QUINLAN, J. R. **C4.5- Programs for Machine Learning**. San Mateo, CA, USA: Morgan Kauffman Publishers, 1993.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning Representations by Backpropagating Errors. **Nature**, [S.l.], v.323, p.533–536, 1986.

SAIDI, F. et al. Online object search with a humanoid robot. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS). **Proceedings...** [S.l.: s.n.], 2007. p.1677–1682.

SCHNEIDER, A. et al. Object identification with tactile sensors using bag-of-features. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS). **Proceedings...** [S.l.: s.n.], 2009. p.243–248.

SCHöLKOPF, B.; SMOLA, A.; MüLLER, K.-R. Nonlinear Component Analysis As a Kernel Eigenvalue Problem. **Neural computation**, [S.l.], v.10, n.5, p.1299–1319, 1998.

SHRIVASTAVA, A. et al. Design of non-linear discriminative dictionaries for image classification. In: **ACCV**. [S.l.]: Springer, 2013. p.660–674.

STRASSEN, V. Gaussian Elimination is not Optimal. **Numerische Mathematik**, Berlin, Germany, v.13, n.3, p.354–356, 1969.

YANG, J.; WANG, J.; HUANG, T. S. Learning the sparse representation for classification. In: INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO (ICME). **Proceedings...** [S.l.: s.n.], 2011. p.1–6.