Unsupervised Deep Learning: Research and Implementation of Variational Autoencoders

Thesis · June 2018

1 author:

Pablo Sánchez Martín
MPI-IS
10 PUBLICATIONS **0** CITATIONS
SEE PROFILE

uc3m | Universidad Carlos III de Madrid

Degree in Telecommunication Technologies Engineering

2017-2018

Bachelor Thesis

Unsupervised Deep Learning

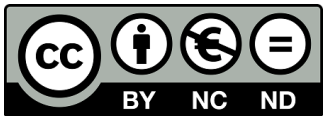
Research and Implementation of Variational Autoencoders

Pablo Sánchez Martín

Supervised by

Pablo Martínez Olmos

Leganés, June 2018



Abstract

Generative models have been one of the major research fields in unsupervised deep learning during the last years. They are achieving promising results in learning the distribution of multidimensional variables as well as in finding meaningful hidden representations in data.

The aim of this thesis is to gain a sound understanding of generative models through a profound study of one of the most promising and widely used generative models family, the variational autoencoders. In particular, the performance of the standard variational autoencoder (known as VAE) and the Gaussian Mixture variational autoencoder (called GMVAE) is assessed. First, the mathematical and probabilistic basis of both models is presented. Then, the models are implemented in Python using the Tensorflow framework. The source code is freely available and documented in a personal GitHub repository created for this thesis. Later, the performance of the implemented models is appraised in terms of generative capabilities and interpretability of the hidden representation of the inputs. Two real datasets are used during the experiments, the MNIST and "Frey faces". Results show the models implemented work correctly, and they also show the GMVAE outweighs the performance of the standard VAE, as expected.

Keywords: deep learning, generative model, variational autoencoder, Monte Carlo simulation, latent variable, KL divergence, ELBO.

ACKNOWLEDGEMENTS

To start with, I would like to thank my family, for being completely supportive throughout the time I have been conducting this thesis and the degree. I would like to thank specially my parents for helping me on a daily basis, both on the good and bad times, for teaching me important principles such as perseverance and basically, for everything they do.

I would also like to thank my university colleagues with which I have shared unforgettable memories during the last years.

Last but not least, I would like to thank my supervisor Pablo M. Olmos. I am extremely grateful to him for the time he has invested in teaching me everything this thesis required, in resolving any issue that arose and for being such a good mentor.

Thank you all for your trust and support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This chapter presents the project in brief. It contains the motivation and objectives of this thesis, the regulatory framework related to the project, the requirements, the activities undertaken and the overall structure of the document.

1.1. Statement of Purpose

Neural networks have existed since the definition of the Perceptron in 1958 [1]. However, they did not become popular until the appearance of backpropagation in the late 80s [2]. Latest advances in distributed computing, specialized hardware and an exponential growth in the amount of data created, have enabled to train extremely complex neural networks with large number of layers and parameters. The discipline that studies this type of networks is known as Deep Learning and it is achieving state-of-the-art results in many difficult problems such as speech recognition [3], object detection [4], text generation [5] and recently in data generation.

A generative model is a kind of unsupervised learning model that aims to learn any type of data distribution and it has the potential to unveil underlying structure in the data. Generative models have recently emerged as a revolution in deep learning with many applications in a wide-ranging variety of fields such as medicine, art, gaming or self-driving. Deep learning, and specially generative models, is evolving so fast that it is difficult to keep up with the latest developments.

For these reasons, the purpose of this work is to **serve as a sound introduction to deep learning generative models**. To achieve this goal, the variational autoencoder model or VAE, one of the most widely used generative models, is studied in detail and compared to one of its variations: the Gaussian Mixture VAE. The implementation of the GMVAE will be based on [6] with some modifications proposed in this work. So the main objective of this bachelor thesis can be expanded as follows:

1. Study in detail the mathematical and probabilistic basis of the variational autoencoders.
2. Explore dimesionality reduction algorithms.
3. Master Tensorflow, a widely used deep learning library for python.
4. Implement the VAE and GMVAE models using different architectures.
5. Compare and evaluate the algorithms implemented for different datasets.

As it will be seen further on, the GMVAE describes a two level hierarchical latent space that aims to capture the multimodal nature of the input data. Due to this, the hypothesis of this thesis is that the GMVAE model improves the generative and clustering capabilities of the standard VAE. Experiments will be conducted to test this hypothesis.

1.2. Requirements

The first part of this thesis is purely theoretical. In the second part, the following tools are required for the implementation of the models: datasets, programming framework and GPU power.

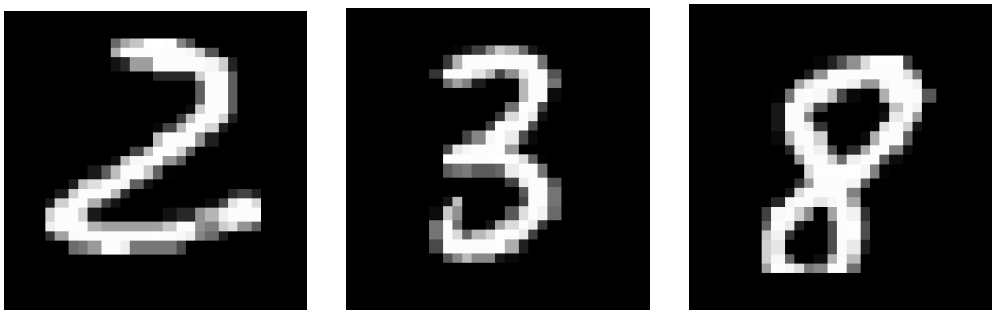
1.2.1. Datasets

The datasets used for this thesis are the MNIST and "Frey faces" (or FREY) datasets, both publicly available. They consist of images with dimension D_x that represents the number of pixels:

$$\text{MNIST: } D_x = 28x28 = 784$$
$$\text{FREY: } D_x = 20x28 = 560$$

MNIST Dataset

The MNIST database [7] is a labeled collection of gray-scale handwritten digits distributed by Yann Lecun's website. Each image is formed by 28x28 size-normalized pixels. The MNIST database is often used to assess performance of algorithms in research. There are functions available to prepare the dataset so there is no need to write code about preprocessing or formatting.



(a) Label: 2 (b) Label: 3 (c) Label: 8

Fig. 1.1. Several samples from the MNIST dataset.

FREY Dataset

The "Fray faces" or FREY dataset [8] consists of 1965 images with 20x28 pixels of Brendan's face taken from sequential frames of a video. Although this is an unlabeled dataset, Brendan appears varying the expressions of his face thus showing different emotions, such as happiness, anger or sorrow, that can be associated to different classes. Several samples are shown in figure 1.2.

(a) Label: Happy (b) Label: Sad (c) Label: Angry

Fig. 1.2. Several samples from Frey faces dataset.

1.2.2. Programming Framework

Python was the programming language chosen to implement the models and run the experiments. Python makes extremely easy to inspect variables and outputs, to debug code and to visualize results. Moreover, there are great tools and software packages for deep learning available in this programming language such as Keras, Theano, TensorFlow or Caffe. In particular, Tensorflow has been used in this thesis.

Tensorflow is an open-source software library developed by Google Brain for internal use. It was released under the Apache 2.0 open source license on November 2015. It provides much control over the implementation and training of neural networks. Moreover, it comes with a wonderful suite of visualization tools known as TensorBoard that allows visualizing TensorFlow graphs, plot the values of tensors during training, visualize embeddings (using T-SNE and PCA algorithms) and much more.

For all these reasons, TensorFlow was the choice to develop this project.

1.2.3. GPU Power

To speed up the training process, the graphic card NVidia GeForce GTX 730 GPU of a personal computer was used to train the models. To utilize the graphic cards of the computer, the GPU implementation of Tensorflow was used. This led to great reduction in training time compared to using the CPU Tensorflow operation mode. However, as the complexity of the models increased, the simulations needed more computational power

to be completed on a reasonable amount of time. Thus, they were launched in the servers of the Signal Processing Group.

1.3. Regulatory Framework

This section discusses the legislation applicable to the realization of this bachelor thesis, namely the development of generative models and the use of personal data. The protection of intellectual property is also presented.

1.3.1. Legislation

Data is an essential tool to train any kind of machine or deep learning algorithm and its quality can condition the resulting model. In some cases the data required is freely available and completely anonymous, for example the MNIST dataset. However, in many other cases the data needed is obtained from users, for example data regarding localization, gender, age or personal interests, and therefore it is necessary to take into account aspects such as privacy, security and ethical responsibility.

On 25th May it came into force in the European Union the General Data Protection Regulation (GDPR) [9], which aims to protect individuals' personal information across the European Union. This new regulation applies to business that somehow use personal data. In short, the GDPR expands the accountability requirements to use personal data, substantially increases the penalties for organizations that do not comply with this regulation and it enables individuals to better control their personal data. The following are the key changings of the GDPR but notice the full document contains 11 chapters and 99 articles:

- Harmonized regularization across the EU.
- Organizations can be fined up to €20 million or 4% of the entity's global gross revenue.
- The individual must clearly consent the use of their personal data by means of "any freely given, specific, informed and unambiguous indication" and this consent must be demonstrable.
- More specific definition of personal data: any direct or indirect information relating to an identified or identifiable natural person.
- Data subjects rights: breach notification, right to be forgotten, access to data collected, data portability, privacy by design and data protection offices.

It is also important to mention that software is a type of Intellectual Property (IP). Therefore, according to the Article I of Spanish Royal Legislative Decree [10], it can be protected by means of copyright, patents or registered trademarks.

1.4. Work Plan

All the activities realized during this bachelor thesis can be grouped in six blocks, as shown in Table 1.1.

Project Documentation	Planification and writing of final report. Document source code in GitHub
Research	Search for papers and articles explaining variational autoencoders and dimensionality reduction algorithms
Datasets selection	Download and test datasets.
Software and hardware selection	Select programming language, deep learning libraries and estimate computation power needed.
Implementation and experimentation	Implement the models and conduct the experiments.
Documents hand-in and presentation	Upload thesis memory and deliver the presentation.

Table 1.1. Work packages.

1.5. Organization

In addition to the introductory chapter, this work contains the following chapters:

- Chapter 2, *Background*, presents the theoretical concepts needed to understand the variational autoencoder and to conduct the experiments along with an overview of generative models.
- Chapter 3, *Variational Autoencoder*, explains in detail the VAE model, describing its architecture and the loss function used for training.
- Chapter 4, *Gaussian Mixture VAE*, explains in detail the GMVAE model, describing its architecture and the loss function used for training.
- Chapter 5, *Methodology*, presents a description of the experiments realized and some important aspects about the implementation of the models.
- Chapter 6, *Results*, shows and explains the results obtained.
- Chapter 7, *Socio-economic environment*, presents some possible practical applications of the study conducted in this work and the socio-economic impact they can generate.
- Chapter 8, *Conclusions*, discusses the results obtained and states the conclusions of the thesis.

- Appendix A, *VAE loss results*, contains a table with all the loss values of the configurations tested.
- Appendix B, *Generated samples*, contains generated samples of VAE.
- Appendix C, *KL Divergence of GMM*, explains why the Kullback-Leibler divergence is intractable when Gaussian mixture models are involved.
- Appendix D, *GMVAE Objective Function Derivations*, explains all the intermediate steps needed to obtain the loss function of the GMVAE model.

2. BACKGROUND

In this section, the theoretical concepts required to understand the VAE are explained. First, a review of the current situation of generative models is presented. Then, the Probabilistic PCA is explained as this algorithm can be understood as a simplified form of the VAE. Thus, this section serves as an introduction to the model in study. Later, a brief introduction of neural networks is made. Finally, a summary of convolutional neural networks, which are really efficient when working with images, is also presented.

2.1. Overview of Generative Models

Generative models are classified as unsupervised learning. In a very general way, a generative model is one that given a number of samples (training set) drawn from a distribution $p(\boldsymbol{x})$, is capable of learning an estimate $p_g(\boldsymbol{x})$ of such distribution somehow.

There are several estimation methods on which generative models are based. However, to simplify their classification [11], only generative models based on maximum likelihood will be considered.

The idea behind the maximum likelihood framework lies in modeling the approximation of the prior distribution of the data, which is known as the likelihood, through some parameters θ: $p_\theta(\boldsymbol{x})$. Then the parameters that maximize the likelihood will be selected.

$$\theta^* = \underset{\theta}{arg\,max} \prod_{i=1}^{N} p_\theta(\boldsymbol{x}^{(i)}) \tag{2.1}$$

In practice, it is typical to maximize the log likelihood $\log p_\theta(\boldsymbol{x}^{(i)})$ as it is less likely to suffer numeric instability when implemented on a computer.

$$\theta^* = \underset{\theta}{arg\,max} \sum_{i=1}^{N} \log p_\theta(\boldsymbol{x}^{(i)}) \tag{2.2}$$

If only generative models based on maximum likelihood are considered , they can be classified in two groups according to how they calculate the likelihood (or an approximation of it): explicit density models or implicit density models. The whole taxonomy tree is shown in figure 2.1.

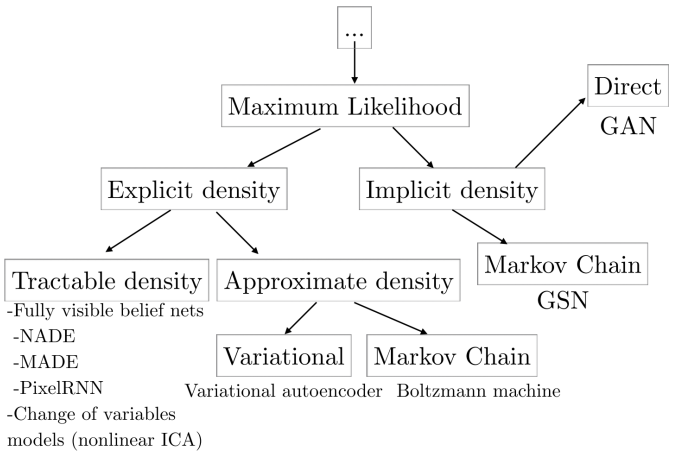


Fig. 2.1. Taxonomy of generative models based on maximum likelihood. Extracted from [11].

2.1.1. Explicit Density Models

Explicit density models can be subdivided in two types: those capable of directly maximizing an explicit density $p_\theta(\boldsymbol{x})$ and those capable of maximizing an approximation of it. This subdivision represents two different ways of dealing with the problem of tractability when maximizing the likelihood. Some of the most famous approaches to tractable density models are Pixel Recurrent Neural Networks (PixelRNN) [12] and Nonlinear Independent Components Analysis (NICA) [13].

The models that maximize an approximation of the density function $p_\theta(\boldsymbol{x})$ are likewise subdivided into two categories depending on whether the approximation is deterministic or stochastic: variational or Markov chain. Later in this work the most widely approach to variational learning, the variational autoencoder also known as VAE will be analyzed in detail (see Chapter 3 and Chapter 4).

Furthermore, Section 2.2 describes one of the simplest explicit density models, Probabilistic PCA, and serves as an introduction to this type of models.

2.1.2. Implicit Density Models

These are models that do not estimate the density probability but instead define a stochastic procedure which directly generates data [14]. This procedure involves comparing real data with generated data. Some of the most famous approaches are the Generative Stochastic Network [15] and Generative Adversarial Network [16].

2.2. Probabilistic PCA

Principal Component Analysis (known as PCA) [17] is a technique used for dimensionality reduction, lossy data compression and data visualization that has wide-ranging applications in fields such as signal processing, mechanical engineering or data analysis. Depending on the field scope, it can also be named discrete Karthunen-Loéve transform.

PCA is a technique that can be analyzed from two different points of view: linear algebra and probability. In this section it is presented an overview of PCA from the probabilistic point of view as it is closely related to the simplest case of the variational autoencoder and a good starting point to latent variable models. The explanation is largely based on Bishop's book [18].

Although PCA was firstly conceived by Karl Pearson in 1901 [19], it was not until the late 90s when probabilistic PCA was formulated by Tipping and Bishop [20] and by Roweis [21]. Probabilistic PCA holds several advantages compared to conventional PCA. One of them is that it can be used as a simple generative model and this is why this algorithm is meaningful in this work.

As its name states, PCA finds the principal components of data, or in other words, it finds the features, the directions where there is the most variance. So given a dataset $\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_1^N$ with dimensionality D_x, the main idea behind PCA is to obtain a subspace (called the principal-component subspace) with dimension D_z, being $D_z << D_x$, so that each $\boldsymbol{x}^{(i)}$ is represented with a $\boldsymbol{z}^{(i)}$ (latent variable) in the best possible way. Then, it is possible to recover $\boldsymbol{x}^{(i)}$ from $\boldsymbol{z}^{(i)}$.

$$\boldsymbol{x}^{(i)} \in \Re^{D_x} \rightarrow \boldsymbol{z}^{(i)} \in \Re^{D_z}$$

The probabilistic PCA is a maximum likelihood solution of a probabilistic latent variable model in which all marginal and conditional distributions are Gaussian. The likelihood function to be maximized is expressed as:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}|z)p(z)dz \tag{2.3}$$

As all distributions are Gaussian, the probability distribution of $\boldsymbol{x}$ can be expressed as $p(\boldsymbol{x}) = N(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{C})$ where the covariance matrix is $C = \boldsymbol{W}\boldsymbol{W}^T + \sigma^2$. Therefore, the parameters of the model are the $D_x x D_z$ matrix $\boldsymbol{W}$, the D_x dimensional vector $\boldsymbol{\mu}$ and the scalar σ^2.

The optimal values of the parameters are obtained using Maximum Likelihood Estimation (MLE), maximizing the log-likelihood $\log p(\boldsymbol{x})$.

$$\log p(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{\mu}, \sigma^2) = \sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)}|\boldsymbol{W}, \boldsymbol{\mu}, \sigma^2)$$

$$= -\frac{D_x N}{2}\log(2\pi) - \frac{N}{2}\log(|\boldsymbol{C}|) - \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{C}^{-1}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu})$$

The optimization can be computed in a closed form [18] or using the EM algorithm [22]. By closed form, the derivative of $\log p(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{\mu}, \sigma^2)$ with respect to each parameter is computed and equal to 0 to obtain the optimal values, namely $\boldsymbol{\mu}_{ML}$, $\boldsymbol{W}_{ML}$ and σ^2_{ML}.

The optimal mean $\boldsymbol{\mu}_{ML}$ is the sample mean of the data:

$$\boldsymbol{\mu}_{ML} = \sum_{i=}^{N} \boldsymbol{x}^{(i)} = \bar{\boldsymbol{x}} \tag{2.4}$$

The expression for the optimal matrix $\boldsymbol{W}$ shown in equation 2.5 is quite complex. However, all the matrices involved are known or can be obtained. The matrix $\boldsymbol{U}_M$ (with shape D_x x D_z) contains the eigenvectors of the data covariance matrix S which correspond to the largest eigenvalues λ_i. These eigenvalues form the diagonal matrix $\boldsymbol{L}_M$, which has shape D_z x D_z. The matrix $\boldsymbol{R}$ (with shape D_z x D_z) is an arbitrary orthogonal matrix.

$$\boldsymbol{W}_{ML} = \boldsymbol{U}_M(\boldsymbol{L}_M - \sigma^2\boldsymbol{I})^{(1/2)}\boldsymbol{R} \tag{2.5}$$

The σ^2_{ML} represents the average variance associated with the discarded dimensions.

$$\sigma^2_{ML} = \frac{1}{D_x - D_z}\sum_{i=D_z+1}^{D_x} \lambda_i \tag{2.6}$$

Now that all the parameters involved are defined and it is known how to compute them, it is possible to define how new samples are generated.

The generation phase comprises two steps. First, the variable $\boldsymbol{z}$ is sample from a priori known distribution $p(\boldsymbol{z})$. Then, new data samples are drawn from $p(\boldsymbol{x}|\boldsymbol{z}) = N(\boldsymbol{x}|\mu(\boldsymbol{z}), \sigma^2 I)$ where $\mu(\boldsymbol{z}) = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}$. This process is shown in figure 2.2.

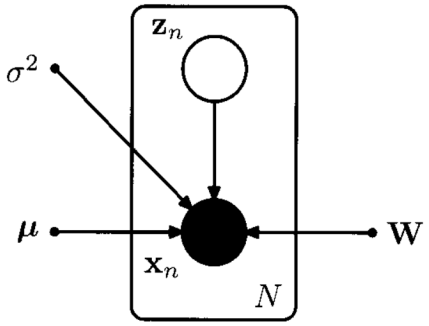


Fig. 2.2. PCA generative model. Extracted from [18].

This is a simple case of a generative model where the relation between the latent and the predictive variables is linear:

$$\boldsymbol{x} \sim \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$$

2.3. Neural Networks

Artificial neural networks are a set of machine learning algorithms that simulates the functioning of biological neurons by means of a mathematical model.

Basically an artificial neuron consists of a set of inputs x_i that are multiplied by weight parameters w_i and added point wise. Then the bias parameter b is added. Finally, the result is introduced into a non-linear function, known as activation function, to obtain the output of the neuron. This mathematical model is shown in figure 2.3 and expressed in equation 2.7.

$$o = \phi\left(\sum_i (x_i w_i) + b\right) \tag{2.7}$$

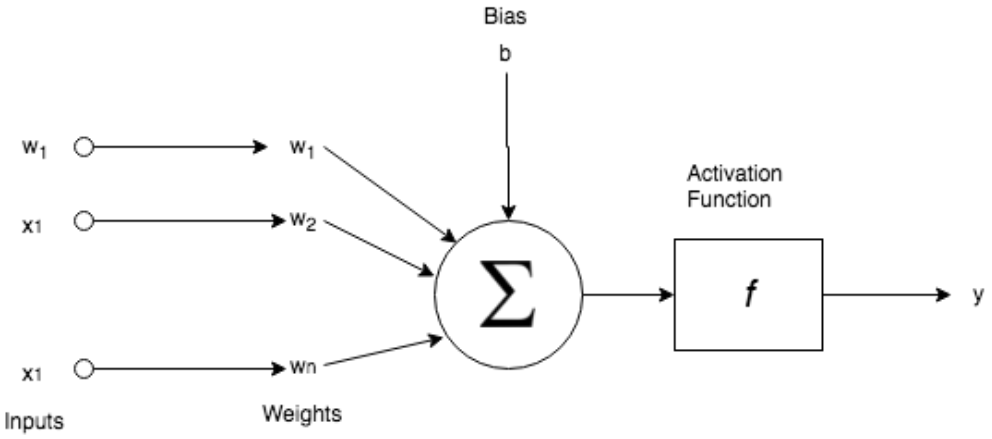
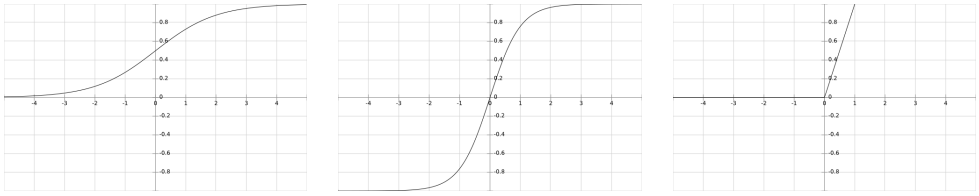


Fig. 2.3. Mathematical model of a simple artificial neuron.

Activation functions (ϕ) are a remarkably important characteristic of neural networks. They determine whether a neuron should be activated or not and the range of possible outputs. In other words, they determine how and how much information flows through the network. There are several functions that can be used as activation functions. The most common ones are identity, ReLU, Leaky ReLU, sigmoid, tahn and softmax. Some of them are shown below in figure 2.4



(a) Sigmoid (b) Tanh (c) ReLU

Fig. 2.4. Several activation functions.

Among them, the Rectifier Linear Unit or ReLU (Equation 2.8) is probably the most widely used function today. It has proven to help to prevent vanishing gradient among other benefits. However, the selection of the activation function depends on the application.

$$\phi(x) = max(0, x) \tag{2.8}$$

Now that all the basic elements of a neural network are described, the question that arise is how to build a neural network. The process is quite simple: one or several neurons form a layer and several stacked layers form a basic neural network. This is probably the most generic architecture of a neural network and it is also the most inefficient. There are different architectures that explote certain characteristics of the data to reduce the required number of parameters to be more efficient. One of them is the convolutional network.

2.3.1. Convolutional Neural Networks

A Convolutional Neural Network (CNN or ConvNet) is a type of feed-forward artificial network that is specifically designed to deal with images. CNNs have proven to be successful in identifying simple and complex objects (i.e faces), in classification tasks and also in providing "vision" capabilities to robots and self-driving cars, among other use-cases.

The theoretical model of the CNN was proposed in the 80s by Yann LeCun [23]. Since then, many modifications of the original model have emerged, and they have enabled to train deeper networks and outweigh previous results. Some of the most common CNNs architectures are AlexNet [24] or recently ResNet [25].

Similarly as with artificial neural networks, CNNs are also inspired in the behaviour of biological neurons. Thus, they are formed by a large amount of artificial neurons with trainable biases and weights. However, the main difference is that these weights are shared in the spatial dimension. This provides CNNs with shift and space invariant properties, which is a very interesting property when dealing with images.

The key component of the CNN is a two-dimensional trainable matrix known as kernel or feature detector. This matrix can be understood as a filter: it traverses an image looking

for certain features or shapes. For instance, in figure 2.5 it is displayed a filter that detects edges.

	0	1	0	
	1	-4	1	
	0	1	0	

(a) Filter Matrix

(b) Result

Fig. 2.5. Edge detector kernel. Extracted from [26].

A convolutional layer consists of a dot product between the input image and one or several kernels. The output of each of these convolutions is known as a feature map. It is possible to find an intuitive visualization of this process in [27]. A set of convolutional layers interwined with pooling (spatial down sampling along the output volume) and activation layers form a CNN. Moreover, it is common to add several fully connected layers at the end of the network to flatten the output. This type of architecture is shown in figure 2.6.

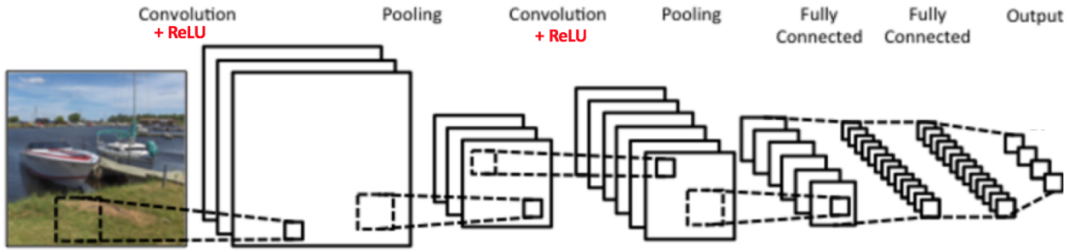


Fig. 2.6. Simple CNN Architecture.

3. VARIATIONAL AUTOENCODER

In this section it is described the variational autoencoder (known as VAE). The explanation starts with a conceptual and general view of how the model works, and subsequently a formal description of its components and objective function.

3.1. Overview

The VAE is a type of generative model that was defined in 2013 [28]. It is a model capable of generating realistic data and obtaining meaningful latent representations of the input. To achieve this, VAE is based on Bayesian inference. VAE models an approximation of the underlying probability distribution of input data $p(\boldsymbol{x})$ through a latent variable z. As mentioned before in this document, VAE can also be understood as a non-linear Probabilistic PCA where the non-linearity is introduced using neural networks.

$$Probabilistic\ PCA: \quad p(\boldsymbol{x}|z) = N(\boldsymbol{W}z + \boldsymbol{\mu}, \sigma^2\boldsymbol{I})$$

$$VAE: \quad p_\theta(\boldsymbol{x}|z) = N(\mu_\theta(z), \sigma^2\boldsymbol{I})$$

Learning $p(\boldsymbol{x})$ can be a remarkably complex task. Data can have large dimensions with complicated relations among them. For example, in the case in which data are gray scale face images, the number of dimensions is *width x height*, namely the number of pixels. For high resolution images it can reach the order of millions. Furthermore, each picture contains borders and complex objects such as eyes, mouth, nose. In addition, each object holds attributes such as orientation, size or shape, and they are located in particular places to form faces. Therefore, it is clear that data has complex structure which the model should learn to be able to generate realistic samples.

So what is a variational autoencoder? Explaining in short, it is an autoencoder, with a random variable z defined in the latent space, that tries to maximize a lower bound of the data log-likelihood.

The VAE is composed of two networks as shown in Figure 3.1 and Figure 3.2. The encoder or inference network (parameterized by ϕ) maps the input data into a latent representation that contains important attributes of the data. This latent representation is then mapped, through the decoder or generative network (parameterized by θ), into the reconstructed data. The main characteristic of VAE is that the latent representation of the input is motivated to be similar to a known distribution. The usual choice is to define the prior distribution $p(z) = N(z|0, \boldsymbol{I})$. In this way, if the latent variable is distributed similarly to $N(0, \boldsymbol{I})$ and it holds the correct information about the input data, it is possible to obtain

samples from z and then generate realistic data.

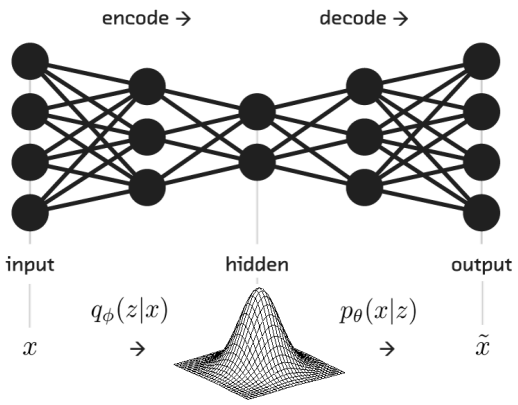


Fig. 3.1. Probabilistic view of VAE. Extracted from [29].

Fig. 3.2. Graphical models for the VAE showing the generative model (left) and the inference model (right).

Before describing the VAE in detail , here are some important points about it:

- Similar inputs will have similar representations in the latent space. So, the latent space will help us interpret input data and their relations.
- The dimension of the latent variable z should be smaller than the dimension of the input so that it captures only the most important features.
- VAE builds an approximation of the true distribution of the input : $p_g(\boldsymbol{x}) \approx p(\boldsymbol{x})$.
- A lower bound of $\log p(\boldsymbol{x})$ called *ELBO* is the functions that will be optimized.
- All density functions involved in the training process are prefixed. In this document, they are assumed to be Gaussian because they will model pixels of images. This way, it is possible to compute its derivatives and use optimization techniques.

3.2. Objective

Variational autoencoders approximately maximize the density function $p(\boldsymbol{x})$ of the training set according to the following formula:

$$p(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|z)p(z)dz \tag{3.1}$$

In an ideal case, the model would perfectly learn the distribution of the data. Therefore, there will be no losses. However, there is one main drawback when solving equation 3.1: it is not possible to have infinite samples of z to compute the integral. Consequently, the equation is intractable and there will be losses.

This section will explain the steps, tricks and assumptions needed to make equation 3.1 tractable and "optimizable". The explanation will derive in the objective function of the VAE, the Evidence Lower Bound (ELBO):

$$\log p(\boldsymbol{x}) \geq \boldsymbol{ELBO}(\theta, \phi, \boldsymbol{x}) = E_{q_\phi(z|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|z)\right] - KL(q_\phi(z|\boldsymbol{x})||p(z)) \tag{3.2}$$

It is important to mention that only a subset of $\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_1^N$ samples from the true distribution $p^*(\boldsymbol{x})$ is available. Then, the marginal log likelihood will be approximated as shown in Equation 3.3, which is an unbiased estimator:

$$E_{p^*(\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x})\right] \approx \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(\boldsymbol{x}^{(i)}) \tag{3.3}$$

To reduce the memory needed to compute it in a computer, only a mini batch of size M ($M << N$) will be considered at a time, the estimator will remain unbiased but with greater variance.

$$E_{p^*(\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x})\right] \approx \frac{1}{M}\sum_{i=1}^{M} \log p_\theta(\boldsymbol{x}^{(i)})$$

Finally, the $ELBO$ will be estimated using M samples $\{\boldsymbol{x}^{(i)}\}_1^M$:

$$\frac{1}{M}\sum_{i=1}^{M} \log p_\theta(\boldsymbol{x}^{(i)}) \geq \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{ELBO}(\theta, \phi; \boldsymbol{x}^{(i)})$$

3.2.1. Dealing with the Integral over z

As previously mentioned, it is necessary to find a formula that can be optimized via gradient descent or similar algorithm [30] in order to maximize $p(\boldsymbol{x})$. The key idea is to approximate the integral with n samples. Let us define equation 3.1 in this way:

$$p(\boldsymbol{x}) = E_{z\sim p(z)}\left[p_\theta(\boldsymbol{x}|z)\right]$$

Then, if it is possible to obtain n samples from $\boldsymbol{z}$, the equation can be written as:

$$p(\boldsymbol{x}) \approx \frac{1}{n}\sum_{i=1}^{n} p_\theta(\boldsymbol{x}|\boldsymbol{z}^{(i)}) \tag{3.4}$$

By convenience, the $\log p_\theta(\boldsymbol{x}|z)$ will be used instead. This will not affect the optimization as the logarithm is a monotonic function[1]. It is assumed $p_\theta(\boldsymbol{x}|z)$ is distributed as a Gaussian $p_\theta(\boldsymbol{x}|z) = N(z|\mu_\theta(z), \Sigma)$. Therefore, its log likelihood is proportional to the euclidean distance between μ and $\boldsymbol{x}$ which is simpler to manipulate analytically.

$$\log p_\theta(\boldsymbol{x}|z) = k - \frac{1}{2}(\boldsymbol{x}-\mu_\theta(z))^T\Sigma^{-1}(\boldsymbol{x}-\mu_\theta(z)) \quad \text{where} \quad k = -\log\left(|\Sigma|^{1/2}(2\pi)^{D_x/2}\right) \quad \Sigma = \sigma^2\boldsymbol{I}$$

The value of σ^2 can be chosen. It is desirable to keep it small as it will be seen later.

3.2.2. Inference Network: $q_\phi(z|\boldsymbol{x})$

According to equation 3.4, the approximation of $p(\boldsymbol{x})$ is better as the number of samples increases. However, the computation power needed also increases. The good news is that in practice many values of $\boldsymbol{z}$ do not generate valid data. In other words, the conditional probability $p_\theta(\boldsymbol{x}|z)$ is close to 0 for the vast majority of $\boldsymbol{z}$. This is why it is defined a family of conditional distributions $q_\phi(z|\boldsymbol{x})$ parameterized by ϕ that represents the distribution of $\boldsymbol{z}$ for each input datum. Thus, the new approximation is:

$$E_{z\sim q_\phi(z|\boldsymbol{x})}\left[\log(p_\theta(\boldsymbol{x}|z)\right]$$

In this way, only values of $\boldsymbol{z}$ that are likely to have produced $\boldsymbol{x}$ are used to compute $p(\boldsymbol{x})$ [31]. We are one step closer to arrive at the final equation.

3.2.3. KL Divergence

The Kullback-Leibler divergence is a measure of how different are two probability functions p and q. One interesting property of the KL divergence which will be really valuable in this study is that it is always a positive number.

For discrete variables it is defines as follows:

$$KL(P(x)||Q(x)) = \sum_{x\in X} P(x)log\frac{P(x)}{Q(x)} \tag{3.5}$$

[1] Function which does not change the maximum

For a continuous variable it is defined as follows:

$$KL(p(x)||q(x) = \int p(x)log\frac{p(x)}{q(x)}dx = E_{x\sim p}[log\frac{p(x)}{q(x)}] \tag{3.6}$$

The model under study only deals with continuous variables thus only equation 3.6 is of interest.

The VAE optimization function will be derived from the idea that $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z}|\boldsymbol{x})$ should be as similar as possible, that is to say, the $KL\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right)$ should be close to 0.

$$KL\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right) = E_{z\sim q_\phi}\left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p(\boldsymbol{z}|\boldsymbol{x})\right]$$

3.2.4. Core Equation

If order to get equation 3.2 it is necessary to recall Bayes Rule:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

Then, applying Bayes rule in $KL\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right)$ and extracting $p(\boldsymbol{x})$ from the expectation because it does not depend on $\boldsymbol{z}$:

$$KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})) = E_{z\sim q_\phi}\left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z})\right] + \log p(\boldsymbol{x}) =$$

$$-E_{z\sim q_\phi}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] + E_{z\sim q_\phi}\left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p(\boldsymbol{z})\right] + \log p(\boldsymbol{x})$$

It is important to notice that the second term is a KL divergence. By rearranging terms, the fundamental equation of the VAE is obtained:

$$\log p(\boldsymbol{x}) - KL\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right) = E_{z\sim q_\phi}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - KL\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right) \tag{3.7}$$

Intuitively this equation tell us that the $\log p(\boldsymbol{x})$ minus an error term, called encoding error, is equal to the mean of $\log p_\theta(\boldsymbol{x}|\boldsymbol{z})$ evaluated at $\boldsymbol{z}$ sampled from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ minus an error term. Also notice the right part looks like an autoencoder: $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ encodes and $p(\boldsymbol{x}|\boldsymbol{z})$ decodes.

The problem of equation 3.7 is that $p(\boldsymbol{z}|\boldsymbol{x})$ is unknown. Thus, it is not possible to calculate $KL\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right)$. This is why this term will be removed from the equation, leading to the loss function of the VAE called *ELBO*:

$$\log p(\boldsymbol{x}) \geq ELBO(\theta, \phi)$$

$$ELBO(\theta, \phi) = E_{z \sim q_\phi} \left[\log p_\theta(\boldsymbol{x}|z)\right] - KL\Big(q_\phi(z|\boldsymbol{x}) \| p(z)\Big) \tag{3.8}$$

The *ELBO* is a lower bound of $\log p(\boldsymbol{x})$. If and only if $KL\Big(q_\phi(z|\boldsymbol{x}) \| p(z|\boldsymbol{x})\Big) = 0$ the *ELBO* is equal to the true distribution $\log p(\boldsymbol{x}) = ELBO(\theta, \phi)$.

It is interesting to describe both terms to understand what they represent:

- The $E_{z \sim q_\phi} \left[\log p_\theta(\boldsymbol{x}|z)\right]$ term represents how good is the reconstruction of the input.
- The $KL\Big(q_\phi(z|\boldsymbol{x}) \| p(z)\Big)$ term is a regularizer. In order to maximize the *ELBO* this term must be small. So this term forces $q_\phi(z|\boldsymbol{x})$ to be similar to $p(z)$, a simple and known distribution, and prevents overfitting.

The next sections develop in detail the loss function and check if it can be optimized through SGD during training

3.2.5. ELBO Optimization

In principle, all the distributions that appear in the *ELBO* are unknown, so they will be assumed to be Gaussian. This assumption is really convenience because it facilitates the computations since the *ELBO* will have a closed form. In addition, it is a meaningful choice from the probabilistic perspective since the Gaussian distribution has the minimal prior structure so it maximizes the entropy [32]. So it is a suitable choice when there is no knowledge about the true nature of a distribution.

Then $q_\phi(z|\boldsymbol{x})$ is defined as:

$$q_\phi(z|\boldsymbol{x}) = N(\mu_\phi(\boldsymbol{x}), \Sigma_\phi(\boldsymbol{x}))$$

And $p(z)$ is defined as:

$$p(z) = N(0, \boldsymbol{I})$$

Both μ and Σ are neural networks with parameters ϕ that are learnt during training. Moreover, Σ is defined as a diagonal matrix so that the $KL\Big(q_\phi(z|\boldsymbol{x}) \| p(z|\boldsymbol{x})\Big)$ has a closed form:

$$KL\Big(N(\mu_\phi(\boldsymbol{x}), \Sigma_\phi(\boldsymbol{x})) \| N(0, \boldsymbol{I})\Big) = \frac{1}{2}\left[tr\Big(\Sigma_\phi(\boldsymbol{x})\Big) - D_z - \log\det(\Sigma_\phi(\boldsymbol{x})) + \Big(\mu_\phi(\boldsymbol{x})\Big)^T \Big(\mu_\phi(\boldsymbol{x})\Big)\right]$$

What about the other term? It involves solving the next integral:

$$E_{z\sim q_\phi}\left[\log p_\theta(\boldsymbol{x}|z)\right] = \int \log p_\theta(\boldsymbol{x}|z) q_\phi(z|\boldsymbol{x}) dz$$

It is approximated using the Monte Carlo estimator which in short it is an unbiased estimator, its variance is inversely proportional to the number of samples $\sigma^2 \propto \frac{1}{\sqrt{M}}$ and if M is large enough the estimated expectation is similar to the real one.

Then, the Monte Carlo estimator of $E_{z\sim q_\phi}\left[\log p_\theta(\boldsymbol{x}|z)\right]$ by taking M samples of z, $(z^{(1)}, ..., z^{(M)})$, from $q_\phi(z|\boldsymbol{x})$ is:

$$E_{z\sim q_\phi}\left[\log p_\theta(\boldsymbol{x}|z)\right] \approx \frac{1}{M}\sum_{i=1}^{M} \log p_\theta(\boldsymbol{x}|z^{(i)})$$

So equation 3.8 can be rewritten as follows:

$$ELBO(\theta,\phi) \approx \frac{1}{M}\sum_{i=1}^{M} \log p_\theta(\boldsymbol{x}|z^{(i)}) - KL\Big(N(\mu_\phi(\boldsymbol{x}), \Sigma_\phi(\boldsymbol{x}))||N(0, \boldsymbol{I})\Big)$$

developing the first term, it has the following form:

$$ELBO(\theta,\phi) = \frac{1}{M}\sum_{i=1}^{M} k - \frac{1}{2}(\boldsymbol{x} - \mu_\theta(z^{(i)}))^T \Sigma^{-1}(\boldsymbol{x} - \mu_\theta(z^{(i)})) - KL\Big(q_\phi(z|\boldsymbol{x})||p(z)\Big) \quad (3.9)$$

where

$$\Sigma = \sigma^2 \boldsymbol{I}_{D_x} \qquad k = -\log\Big(|\Sigma|^{1/2}(2\pi)^{D_x/2}\Big) \qquad |\Sigma| = (\sigma^2)^{D_x}$$

From equation 3.9, it is important to notice that σ^2 acts as a weighting factor between the two terms. However, take into account that the existence of this parameter depends on the choice of $p_\theta(\boldsymbol{x}|z)$. Our choice of σ determines how accurately we expect the model to reconstruct $\boldsymbol{x}$. If σ^2 is small, the first term outweighs the second one forcing $\boldsymbol{x} \approx \mu_\theta(z)$.

Now, this formula looks better, more tractable. However, can it be optimized via backpropagation? The answer is no because it is necessary to obtain z samples in an intermediate layer to calculate $\log p_\theta(\boldsymbol{x}|z^{(i)})$:

$$\boldsymbol{x} \rightarrow \mu_\phi, \Sigma_\phi \rightarrow z \sim N(\mu_\phi, \Sigma_\phi) \rightarrow \mu_\theta \rightarrow \boldsymbol{x}$$

Because of this, the forward pass of this network works fine and, if the output is averaged over many samples of $\boldsymbol{x}$ and z, it produces the correct expected value. Nevertheless, it is not possible to back-propagate the error through stochastic layers. Sampling is a non-continuous operation and has no gradient. In order to backpropagate the only source of stochastic can be the inputs. This is what "Reparametrization Trick" achieves.

3.2.6. Reparameterization Trick

In order to obtain the latent representation z of a given observation $\boldsymbol{x}$, it is necessary to sample it from the approximate posterior $q_\phi(z|\boldsymbol{x})$. The reparameterization trick consists in moving the sampling from an inner layer to an input layer. This is achieved defining a new input ϵ that introduces randomness into the model. Now, rather than sampling $z \sim N(\mu_\phi, \Sigma_\phi)$ directly, the sampling is done in $\epsilon \sim N(0, \boldsymbol{I})$ and then z is calculated in a deterministic way $z = \mu_\phi(\boldsymbol{x}) + \epsilon \sqrt{\Sigma_\phi(\boldsymbol{x})}$:

$$\boldsymbol{x}, \epsilon \rightarrow \mu_\phi, \Sigma_\phi \rightarrow z = \mu_\phi + \sqrt{\Sigma_\phi} * \epsilon \rightarrow \mu_\theta \rightarrow \boldsymbol{x}$$

In this way, backpropagation can be applied to optimize the network parameters. So now:

$$ELBO(\theta, \phi) = \frac{1}{M} \sum_{i=1}^{M} \log p(\boldsymbol{x}|z^{(i)} = \mu_\phi(\boldsymbol{x}) + \epsilon^{(i)} \sqrt{\Sigma_\phi(\boldsymbol{x})}) - KL\left(q_\phi(z|\boldsymbol{x})||p(z)\right) \tag{3.10}$$

If $\boldsymbol{x}$ and ϵ are fixed, Equation 3.10 is deterministic and continuous in θ and ϕ. This means it is possible to use SGD to optimize it via backpropagation. Finally, the *ELBO* is tractable and can be optimized.

3.3. Sampling

Now that every part of the model has been explained, the sampling of new data is quite straightforward: z is sampled from its prior distribution $p(z) = N(0, \boldsymbol{I})$ and injected into the decoder:

$$z \sim N(0, \boldsymbol{I}) \rightarrow \mu_\theta(z) \rightarrow \boldsymbol{x}$$

4. GAUSISIAN MIXTURE VARIATIONAL AUTOENCODER

4.1. Introduction

The Gaussian Mixture variational autoencoder (known as GMVAE) is a modification of the standard VAE that adds structure to the latent space so that it is able to capture the multimodal nature of the data. This model can lead to a better interpretability of the latent space (better unsupervised clustering) and better reconstruction and generative capabilities compared to VAE. During the last years several ideas have been proposed to implement the GMVAE. Examples of these proposals can be found in [33], [6] or [34].

4.2. Overview

The description of the GMVAE model described in this work will be largely based on model presented by Nat Dilokthanakul et al [6]. Nevertheless, the definition of the prior conditional term of the objective function (explained later) derived in this work differs from the one proposed in the aforementioned paper. As with VAE, GMVAE consists of two clearly differentiated parts: the inference and the generative networks. Nevertheless, as can be seen in figure 4.1 there are more variables at stake.

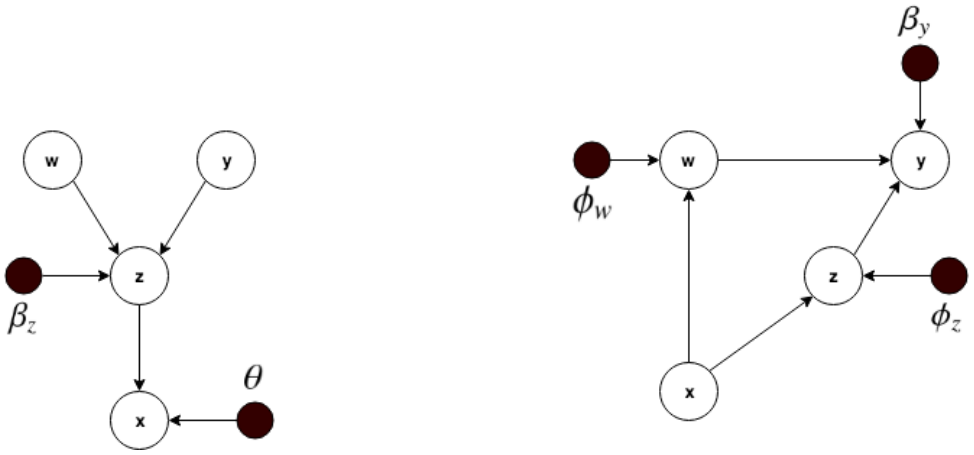


Fig. 4.1. Graphical models for the GMVAE showing the generative model (left) and the inference model (right).

The generative model shown in figure 4.1 provides a good intuition of what represents each variable. The idea is that the latent variable y represents the class of the data to be generated and the latent variable $\boldsymbol{w}$ captures variable attributes of the data. These two variables form the first latent hierarchical level. Finally, in the second latent hierarchical level is the variable $\boldsymbol{z}$. This variable will have the form of a Gaussian mixture distribution

and it will be a representation of the multimodal nature of the data. Now that the intuitive meaning of each variable is clear, their definitions will make sense:

- Input data: $\boldsymbol{x} \sim p(\boldsymbol{x})$
- Latent variable $\boldsymbol{z}$: $p(\boldsymbol{z}) = \int p(\boldsymbol{z}|\boldsymbol{w}, y)p(\boldsymbol{w})p(y)d\boldsymbol{w}dy$
- Latent variable $\boldsymbol{w}$: $p(\boldsymbol{w}) = N(0, \boldsymbol{I})$
- Latent variable y: $y \sim Mult(\boldsymbol{\pi})$ given that $\pi_i = \frac{1}{K}$

4.3. Defining z directly as a GMM

The most direct solution to inject multi-modality into the latent space would be to use the standard VAE framework but defining $\boldsymbol{z}$ as a Gaussian mixture density.

$$p(\boldsymbol{z}) = \sum_{i=1}^{K} \Pi_i N(\mu_i, \Sigma_i)$$

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = N(\boldsymbol{z}|\mu_\phi(\boldsymbol{x}), \Sigma_\phi(\boldsymbol{x}))$$

However, the problem with this approach is that using GMM makes the Kullback-Leibler divergence of the *ELBO* equation 3.8 intractable (see Appendix C). While it is possible to approximate it using Monte Carlo estimator, this increases the variance of the estimator.

$$KL\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\Big) = E_{\boldsymbol{z}\sim q_\phi}\left[\log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z})}\right] \approx \frac{1}{M}\sum_{i=1}^{M} \log \frac{q_\phi(\boldsymbol{z}^{(i)}|\boldsymbol{x})}{p(\boldsymbol{z}^{(i)})}$$

Another problem with this approach is that it produces numerical instability during training when $q_\phi(\boldsymbol{z}^{(i)}|\boldsymbol{x}) \to 0$ or $p(\boldsymbol{z}^{(i)}) \to 0$ or both go to 0 (see section 5.6 for more information). Thus, this approach is discarded.

4.4. Optimization Function

As stated in the paper, the objective function (known as ELBO) is defined in the following manner:

$$ELBO = E_q[\frac{p_{\beta_z,\theta}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w}, y)}{q_{\beta_y,\phi_z,\phi_w}(\boldsymbol{z}, \boldsymbol{w}, y|\boldsymbol{x})}]$$

The definitions of $p_{\beta_z,\theta}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w}, y)$ and $q_{\beta_y,\phi_z,\phi_w}(\boldsymbol{z}, \boldsymbol{w}, y|\boldsymbol{x})$ can be arbitrarily chosen. For convenience, they are defined as shown in equation 4.1 and equation 4.2 respectively.

From these definitions, it is important to remember that the parameters of each distribution means it is implemented using neural networks.

$$p_{\beta_z,\theta}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w}, y) = p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_{\beta_z}(\boldsymbol{z}|\boldsymbol{w}, y) p(\boldsymbol{w}) p(y) \tag{4.1}$$

$$q_{\beta_y,\phi_z,\phi_w}(\boldsymbol{z}, \boldsymbol{w}, y|\boldsymbol{x}) = \prod_i p_{\beta_y}(y^{(i)}|\boldsymbol{w}^{(i)}, \boldsymbol{z}^{(i)}) q_{\phi_z}(\boldsymbol{z}^{(i)}|\boldsymbol{x}^{(i)}) q_{\phi_w}(\boldsymbol{w}^{(i)}|\boldsymbol{x}^{(i)}) \tag{4.2}$$

The join density function $p_{\beta_z,\theta}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w}, y)$ is decomposed in a multiplication of four terms. The prior density functions $p(\boldsymbol{w})$ and $p(y)$ are known (see section 4.3). The conditional distribution of $\boldsymbol{z}|\boldsymbol{w}, y$ is defined as $p_{\beta_z}(\boldsymbol{z}|\boldsymbol{w}, y) = \prod_{k=1}^{K} N(\mu_{\beta_z}(\boldsymbol{w}), \Sigma_{\beta_z}(\boldsymbol{w}))^{[y_k==1]}$ being K the number of clusters selected a priori. Finally, the conditional distribution of $\boldsymbol{x}|\boldsymbol{z}$ is defined as $p_\theta(\boldsymbol{x}|\boldsymbol{z}) = N(\mu_\theta(\boldsymbol{z}), \Sigma)$.

As it is done in the original paper, to simplify the notation only one data point at a time will be considered. Then, the conditional distribution $q_{\beta_y,\phi_z,\phi_w}(\boldsymbol{z}, \boldsymbol{w}, y|\boldsymbol{x})$ is decomposed in a multiplication of three terms. The conditional distributions of $\boldsymbol{z}|\boldsymbol{x}$ and $\boldsymbol{w}|\boldsymbol{x}$ are both defined as Gaussian being $q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x}) = N(\mu_{\phi_w}(\boldsymbol{x}), \Sigma_{\phi_w}(\boldsymbol{x}))$ and $q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x}) = N(\mu_{\phi_z}(\boldsymbol{x}), \Sigma_{\phi_z}(\boldsymbol{x}))$ respectively. The distribution $p_{\beta_y}(y_j = 1|\boldsymbol{w}^{(i)}, \boldsymbol{z}^{(i)})$ is parameterized by β_y because during the experiments it will be modeled using a neural network to assure numerical stability (see section 5.6). However, the most accurate definition of such distribution is:

$$p_{\beta_z}(y_j = 1|\boldsymbol{w}, \boldsymbol{z}) = \frac{p(y_j = 1) p_{\beta_z}(\boldsymbol{z}|y_j = 1, \boldsymbol{w})}{\sum_{k=1}^{K} p(y_k = 1) p_{\beta_z}(\boldsymbol{z}|y_k = 1, \boldsymbol{w})}$$

Given this definition of the distributions and considering $E_q \equiv E_{q_{\beta_y,\phi_z,\phi_w}(\boldsymbol{z},\boldsymbol{w},y|\boldsymbol{x})}$, the $ELBO$ function can be rewritten as:

$$ELBO = E_q\left[\frac{p(\boldsymbol{w})p(y)p_{\beta_z}(\boldsymbol{z}|\boldsymbol{w}, y)p_\theta(\boldsymbol{x}|\boldsymbol{z})}{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})p_{\beta_z}(y|\boldsymbol{w}, \boldsymbol{z})}\right] =$$

$$E_q\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] + E_q\left[\log \frac{p_{\beta_z}(\boldsymbol{z}|\boldsymbol{w}, y)}{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\right] + E_q\left[\log \frac{p(\boldsymbol{w})}{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}\right] + E_q\left[\log \frac{p(y)}{p_\beta(y|\boldsymbol{w}, \boldsymbol{z})}\right] =$$

$$E_q\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - E_q\left[\log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_{\beta_z}(\boldsymbol{z}|\boldsymbol{w}, y)}\right] - E_q\left[\log \frac{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}{p(\boldsymbol{w})}\right] - E_q\left[\log \frac{p_{\beta_z}(y|\boldsymbol{w}, \boldsymbol{z})}{p(y)}\right]$$

Each of the terms composing the $ELBO$ has a specific function. In order, they are named reconstruction term, conditional prior term, w-prior term and z-prior term. In the following sections each of these terms is operated and simplified as much as possible. The complete steps are shown in Appendix D.

4.4.1. Reconstruction Term

The first component is the reconstruction term. It measures the difference between the input data and their reconstruction. This term has exactly the same form as the first term of the *ELBO*'s VAE.

$$E_q[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] \approx \frac{1}{2\sigma^2}(\boldsymbol{x} - \mu_\theta(\boldsymbol{z}))^2$$

4.4.2. Conditional prior term

The conditional prior term $E_q\left[\log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y)}\right]$ measures the similarity between $q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})$ and $p_\beta(\boldsymbol{z}|\boldsymbol{w},y)$. As it is a positive value and appears with a negative sign in the *ELBO* equation, the conditional prior term should be small in order to maximize the objective function. To be a small value, both probability functions should be similar to each other. Thus, this term is restricting the expressive capacity of the model: it is a regularizer.

In the model presented by Nat Dilokthanakul et al [6], this term is simplified and results in an expectation of a KL divergence. Then it is approximated using Monte Carlo.

$$\begin{aligned} &E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})p_\beta(y|\boldsymbol{w},\boldsymbol{z})}\left[KL\Big(q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})\|p_\beta(\boldsymbol{z}|\boldsymbol{w},y)\Big)\right] \\ &\approx \frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K} p_\beta(y_k=1|\boldsymbol{w}^{(i)},\boldsymbol{z}^{(i)})KL\Big(q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})\|p_\beta(\boldsymbol{z}|\boldsymbol{w}^{(i)},y_k=1)\Big) \end{aligned}$$

To do the approximation, the latent variable $\boldsymbol{w}$ is sampled from $q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})$ but it is not clear how $\boldsymbol{z}$ is sampled to obtain the final result. For this reason, the simplified expression for this term has been derived in this thesis and it turns out it differs from the one proposed in the aforementioned paper.

$$E_q\left[\log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y)}\right] = E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^{K}\tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right]$$

As it can be observed, now it is clear this term can be approximated using the Monte Carlo estimator by sampling $\boldsymbol{w}$ from $q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})$ and $\boldsymbol{z}$ from $q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})$:

$$E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^{K}\tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right] \approx \frac{1}{N}\frac{1}{M}\sum_{n=1}^{N}\sum_{m=1}^{M}\sum_{k=1}^{K}\tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}^{(m)}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}^{(m)}|\boldsymbol{w}^{(n)},y_k=1)}$$

This expression can be simplified considering only one sample for each variable.

$$E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^{K}\tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right] \approx \sum_{k=1}^{K}\tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}$$

Moreover, as all variables involved are distributed as Gaussians, it can be further simplified. For the complete details see Appendix D.

4.4.3. W-prior term

The w-prior term $E_q\left[\log \frac{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}{p(\boldsymbol{w})}\right]$ measures the similarity between $q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})$ and $p(\boldsymbol{w})$. As it is a positive value and appears with a negative sign in the *ELBO* equation, the w-prior term is also a regularizer. It can be written as a KL divergence of Gaussian distributions. Consequently, it can be computed analytically.

$$E_q\left[\log \frac{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}{p(\boldsymbol{w})}\right] = KL\left(q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})||p(\boldsymbol{w})\right)$$

Recalling that $p(\boldsymbol{w}) = N(0, \boldsymbol{I})$:

$$KL\left(q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})||p(\boldsymbol{w})\right) = \frac{1}{2}\left[tr\left(\Sigma_{\phi_w}(\boldsymbol{x})\right) - D_w - \log\det(\Sigma_{\phi_w}(\boldsymbol{x})) + \left(\mu_{\phi_w}(\boldsymbol{x})\right)^T\left(\mu_{\phi_w}(\boldsymbol{x})\right)\right]$$

4.4.4. Y-prior term

The y-prior term $E_q\left[\log \frac{p_\beta(y|\boldsymbol{w},\boldsymbol{z})}{p(y)}\right]$ measures the similarity between $p_\beta(y|\boldsymbol{w},\boldsymbol{z})$ and $p(y)$. As it is a positive value and appears with a negative sign in the *ELBO* equation, the y-prior term is also a regularizer. It can be written as an expectation over $q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})$ and $q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})$ of a KL divergence of categorical distributions.

$$E_q\left[\log \frac{p_\beta(y|\boldsymbol{w},\boldsymbol{z})}{p(y)}\right] = E_{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}\left[KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right)\right]$$

$$KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right) = \sum_{i=1}^{K} p(y_i=1)\log\frac{p(y_i=1)}{p_\beta(y_i=1|\boldsymbol{w},\boldsymbol{z})}$$

Recalling that $p(y_i = 1) = \frac{1}{K}$

$$KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right) = -\log K - \frac{1}{K}\sum_{i=1}^{K}\log p_\beta(y_i=1|\boldsymbol{w},\boldsymbol{z})$$

4.4.5. ELBO Equation

Joining the previous expressions, the lower bound function can be written as:

$$ELBO(\phi_w,\phi_z,\beta,\theta) = E_{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^{K}\tilde{\pi}_k\log\frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right] - KL\left(q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})||p(\boldsymbol{w})\right) - E_{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}\left[KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right)\right] \tag{4.3}$$

This is the expression that will be optimized during training, having applied reparameterization trick previously.

4.5. Reparameterization Trick

As it happens in the VAE, it is necessary to move the intermediate sampling layers to the input in order to be able to apply backpropagation. Therefore, the sampling from $q_{\phi_z}(z|\boldsymbol{x})$ and $q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})$ is realized through two new input noises, $\epsilon_z \sim N(0, \boldsymbol{I})$ and $\epsilon_w \sim N(0, \boldsymbol{I})$, as shown below:

$$z = \mu_{\phi_z}(\boldsymbol{x}) + \epsilon \sqrt{\Sigma_{\phi_z}(\boldsymbol{x})} \sim q_{\phi_z}(z|\boldsymbol{x}) = N(z|\mu_{\phi_z}(\boldsymbol{x}), \Sigma_{\phi_z}(\boldsymbol{x}))$$

$$w = \mu_{\phi_w}(\boldsymbol{x}) + \epsilon \sqrt{\Sigma_{\phi_w}(\boldsymbol{x})} \sim q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x}) = N(\boldsymbol{w}|\mu_{\phi_w}(\boldsymbol{x}), \Sigma_{\phi_w}(\boldsymbol{x}))$$

In this way, backpropagation can be applied to optimize the network parameters since given a fixed $\boldsymbol{x}$, ϵ_z and ϵ_w, the $ELBO$ is deterministic and continuous, meaning it is possible to use SGD to optimize it via backpropagation.

4.6. Sampling

The sampling of new data is straightforward. First, $\boldsymbol{w}$ and y are sampled from their corresponding prior distributions $p(\boldsymbol{w}) = N(0, \boldsymbol{I})$ and $p(y) \sim Mult(\boldsymbol{\pi})$. Then $\boldsymbol{z}$ is sampled from $p_{\beta_z}(z|\boldsymbol{w}, y)$ and injected into the decoder:

$$\boldsymbol{w} \sim N(0, \boldsymbol{I}) \quad , \quad y \sim Mult(\boldsymbol{\pi}) \rightarrow \boldsymbol{z} \sim p_{\beta_z}(z|\boldsymbol{w}, y) \rightarrow \mu_\theta(\boldsymbol{z}) \rightarrow \boldsymbol{x}$$

5. METHODOLOGY

This chapter presents everything related to the experiments conducted. It includes the objectives, the preparation of the data, the definition of the models' architectures and hyperparameters, the problems found during training, the definitions of the metrics used to evaluate the results and finally some aspects related to the visualization of the results.

5.1. Objective

The purpose of the experiments conducted in this work is to understand the effect of different configurations of the models' hyperparameters and to assess if the GMVAE described in chapter 4 achieves superior performance, in terms of generative capability and latent space interpretability, compared to the standard VAE.

5.2. Prepare Data

Both datasets used in this thesis are already cleaned and contain no missing data. They will be firstly split in test and train. Then the train set will be divided in two subsets as shown in Figure 5.1 to obtain a validation set. Thus, three subsets of images will be used to realize the experiments, each of them with a specific purpose:

- Training Data: data used in the learning process.
- Validation Data: data used to evaluate the loss in each epoch during training. It is not used in the learning process but to stop training earlier to prevent overfitting (stop training when validation error increases)
- Test Data: data used to generate results and assess the performance of the model.

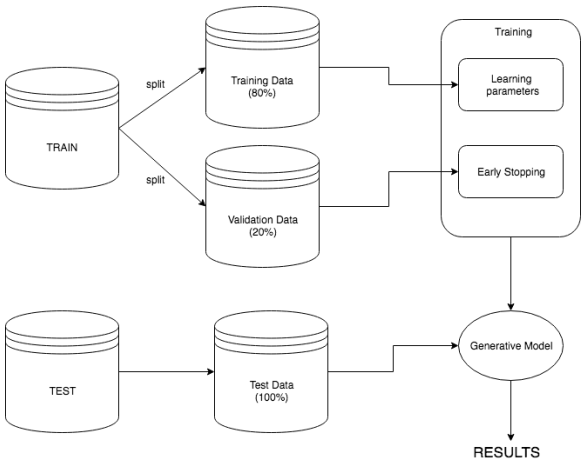


Fig. 5.1. Data diagram.

5.3. Model Architecture

Three different types of neural networks have been used to build the models:

- DenseNet: Formed by simple fully connected layers.
- ConvNet: Formed by convolutional layers, pooling layers and a DenseNet at the end.
- DeconvNet: Formed by deconvolutional layers and a DenseNet at the beginning.

Then, with these three types of neural networks two types of architectures have been used to implement VAE and GMVAE. They have been named *Dense Architecture* and *Conv Architecture* . Table 5.1 and Table 5.2 show how each distribution has been implemented for each architecture.

Distribution	Dense Architecture	Conv Architecture
$q_\phi(\boldsymbol{z}\|\boldsymbol{x})$	DenseNet	ConvNet
$p_\theta(\boldsymbol{x}\|\boldsymbol{z})$	DenseNet	ConvNet

Table 5.1. VAE architectures

Distribution	Dense Architecture	Conv Architecture
$p_{\beta_z}(\boldsymbol{z}\|\boldsymbol{w}, y)$	DenseNet	DenseNet
$p_\theta(\boldsymbol{x}\|\boldsymbol{z})$	DenseNet	DeconvNet
$p_{\beta_y}(y\|\boldsymbol{w}, \boldsymbol{z})$	DenseNet	DenseNet
$q_{\phi_w}(\boldsymbol{w}\|\boldsymbol{x})$	DenseNet	ConvNet
$q_{\phi_z}(\boldsymbol{z}\|\boldsymbol{x})$	DenseNet	ConvNet

Table 5.2. GMVAE architectures

5.4. Definition of hyperparameters

In general terms, when training a machine or deep learning algorithm there are a number of parameters that can be selected before training. Some of them handle the files generated, others deal with saving and restoring a model and others determine certain aspects of the algorithm. In order to test different configurations of the VAE and GMVAE algorithms, some parameters must variate. These are defined in Table 5.3.

Hyperparameter	Definition
Epochs	The number of epochs determines how many times each training image is used during training.
dim z	The z dimension defines the dimension of the latent variable z.
dim w	The w dimension defines the dimension of the latent variable w. Used only in GMVAE
K clusters	The K clusters set the number of possible values that the categorical variable y can take. Used only in GMVAE
Hidden dimension	The hidden dimension specify the number of neurons in a single dense layer.
Number of layers	The number of layers determines the layers used in a dense network.

Table 5.3. Definition of hyperparameters.

5.5. Experimental Setup

To accomplish the goals defined previously, the following experiments were conducted:

- Experiment 1. Train VAE using different configuration of the hyperparamters (shown in Table 5.4). Evaluate the results using the metrics defined in Section 5.8. Compare the latent space representation and input's reconstruction with the results PCA achieves.
- Experiment 2. Train GMVAE changing dim w (see Table 5.5) to understand the information learnt by the latent variables w and z. Evaluate the results in terms of interpretability.
- Experiment 3. Train GMVAE using the best configurations obtained in experiment 1 in terms of generative and reconstruction capabilities. As the dataset consists of handwritten numbers from 0 to 9, it is known the number of classes is 10, therefore $K = 10$. Evaluate the results using the metrics defined in Section 5.8.
- Experiment 4. Evaluate the log likelihood of VAE and GMVAE models trained in previous experiments when uniform and dropout noise is added to the input. The uniform noise (soft) adds ± 0.1 to each normalized pixel and the dropout noise (strong) sets to 0 half of the pixels.
- Experiment 5. Train GMVAE and VAE using convolutional and deconvolutional networks. Compare the results with respect to the ones obtained in experiment 3 and experiment 1. Evaluate performance using "Frey faces" dataset. Configuration is shown in Table 5.6.

All models have been trained using images as input. As it is known, the most efficient way to deal with images is to use convolutional layers. However, simple fully connected dense networks have been used for the first four experiments in order to minimize training time.

The selection of the set of values for each hyperparameter was based on hyperparameters selected in similar works and on interpretability.

Experiment 1	
Generative model	VAE
Model name	model1
Dataset	MNIST
Architecture	Dense Architecture
σ^2	[0.01, 0.001]
$\dim z$	[2, 10, 20, 50]
Hidden layers	[2, 3, 4]
Hidden dimension	[64, 128]

Table 5.4. Experiment 1 Hyper-Parameters.

Experiment 2	
Generative model	GMVAE
Model name	model4_bias
Dataset	MNIST
Architecture	Dense Architecture
σ^2	0.001
$\dim z$	10
$\dim w$	[2, 10, 20]
K	10
Hidden layers	3
Hidden dimension	128

Table 5.5. Experiment 2 Hyper-Parameters.

Experiment 5	
Generative model	[GMVAE, VAE]
Model name	[model5, model2]
Dataset	[MNIST, FREY]
Architecture	Conv Architecture

Table 5.6. Experiment 5 Hyper-Parameters.

Next section describes some problems encountered during training.

5.6. GMAVE numerical stability

In the first attempt to implement the GMVAE, numerical stability problems were found. At some point during training (usually at the beginning), NaN values appeared in the loss function. As a consequence, training stopped.

After a sound analysis, it was found the numerical instability was originated in the evaluation of the distribution $p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z})$:

$$p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z}) = \frac{p(y_j = 1)p_\beta(\boldsymbol{z}|y_j = 1, \boldsymbol{w})}{\sum_{k=1}^{K} p(y_k = 1)p_\beta(\boldsymbol{z}|y_k = 1, \boldsymbol{w})} = \frac{\tilde{\pi_j} N(\boldsymbol{z}|\mu_\beta(\boldsymbol{w})^{y_j}, \sigma_\beta^2(\boldsymbol{w})^{y_j})}{\sum_{k=1}^{K} \tilde{\pi_k} N(\boldsymbol{z}|\mu_\beta(\boldsymbol{w})^{y_k}, \sigma_\beta^2(\boldsymbol{w})^{y_k})}$$

The problem arose when evaluating $\boldsymbol{z}$ in $p_\beta(\boldsymbol{z}|y_j = 1, \boldsymbol{w})$, which is assumed to be a Gaussian distribution. If $\boldsymbol{z}$ is at great distance from the mean of the distribution, the probability is approximately 0. On the other hand, if the variance learnt tends to 0 or ∞ problems arise. In these cases, the evaluation of the probabilities $p_\beta(\boldsymbol{z}|y_k = 1, \boldsymbol{w})$ or $p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z})$ may be of indeterminate form, for example $\frac{0}{0}$ or $0x\infty$, which causes the program to throw a *NaN* value.

In order to solve this problem and reach a robust implementation, three complementary solutions have been approached:

- Change the definition of $p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z})$
- Restrict the output of the neural networks related to variables $\boldsymbol{z}$ and $\boldsymbol{w}$.
- Reduce the learning rate

5.6.1. Approximation of $p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z})$

As described previously, the direct evaluation of $p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z})$ leads to numerical instability. Thus, the following alternatives were proposed and tested.

Softmax approximation

The first idea was to approximate $p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z})$ with a softmax function. Nevertheless, it also had numerical stability issues.

$$\boldsymbol{v}_j = \boldsymbol{z} - \mu_\beta(\boldsymbol{w})^{y_j}$$
$$p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z}) = softmax(\boldsymbol{v}_j) = \frac{e^{\boldsymbol{v}_j}}{\sum_{k=1}^{K} e^{\boldsymbol{v}_k}}$$

As explained in the next section, the problem was mitigated by limiting the range values of several neural networks outputs. However, the limitations were so strong that the model performed poorly. This alternative was discarded.

Constant PDF

The next alternative was to simplify the model by defining $p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z})$ as :

$$p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z}) = \frac{1}{K} \forall j$$

This way, it is not required to evaluate the probabilities $p_\beta(\boldsymbol{z}|y_k = 1, \boldsymbol{w})$ so, obviously, the instability problem disappears. However this is a strong assumption that reduces the expressivity of the model. Thus, further alternatives were sought.

Neural network approximation

The final approach was to approximate the PDF using a neural network composed of fully connected layers. The input of this neural network is the concatenation of $\boldsymbol{w}$ and $\boldsymbol{z}$ and the output is the probability for each y_j.:

$$p_\beta(y_j = 1|\boldsymbol{w}, \boldsymbol{z}) = q_{\beta_y}(y_j = 1|[\boldsymbol{w}, \boldsymbol{z}])$$

This resulted in a robust training as the numerical instability problem disappeared: there is no need to evaluate $p_\beta(\boldsymbol{z}|y_k = 1, \boldsymbol{w})$. Moreover, the dependency of y on $\boldsymbol{z}$ and $\boldsymbol{w}$ still remains and the y-prior term of the *ELBO* prevents overfitting on this neural network.

5.6.2. Output restrictions

As described above, the instability problem arise when the variable $\boldsymbol{z}$ must be evaluated at distant points from the mean of the distribution or its variance tends to 0 or ∞. However this situation is nonsense as every image should have a latent representation that is related to one cluster and extreme values of the parameters of the Gaussian distributions are not interpretable. This is the reason to impose constrains in the PDFs $p_{\beta_z}(\boldsymbol{z}|\boldsymbol{w}, y)$ and $q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})$. In other words, to impose restrictions in the output range of the following neural networks:

- $\mu_{\phi_z}(\boldsymbol{x})$, $\mu_{\beta_z}(\boldsymbol{w})^{y_j}$

- $\sigma^2_{\phi_z}(\boldsymbol{x})$, $\sigma^2_{\beta_z}(\boldsymbol{w})^{y_j}$

To restrict the possible values of outputs of the neural networks modeling the means and the variances it has been used $\phi(\boldsymbol{x}) = A * tf.tanh(\boldsymbol{x})$, where $A \in [1, 100]$, as the activation function of the last layer of the neural networks. Moreover, as the variance cannot be negative neither close to zero, the following sigma function was defined, being $\boldsymbol{x}$ the output of the neural network:

$$\sigma^2(\boldsymbol{x}) = e^{\boldsymbol{x}} \rightarrow \sigma^2(\boldsymbol{x}) = \log(e^{\boldsymbol{x}} + 1) + 0.1 = tf.softplus(\boldsymbol{x}) + 0.1$$

With this definition of the variance and the constrains in the range of outputs the problem is solved.

5.7. Motivate clustering behaviour

Other problem found is anti-clustering behaviour. As it is stated in [6], the z-prior term can degenerate the clustering nature of the latent variable $\boldsymbol{z}$. It is a regularization term that may go to 0 if all the clusters merge into a single one. This over-regularization issue is an open problem under research.

To stimulate clustering behaviour, the bias for the last layer of the neural networks $\mu_{\phi_z}(\boldsymbol{x})$ and $\mu_{\beta_z}(\boldsymbol{w})^{y_j}$ is not initialize with zeros (as usual) but as a truncated normal. We are just providing the model with a priori knowledge to facility the learning process.

5.8. Metrics

The selection of metrics to evaluate the performance of generative models is still an open discussion in spite of the fact that there has been significance progress. In general terms, a good metric should:

- Measure the visual interpretation of samples (subjective).
- Maximize the log-likelihood.

It would be perfect to have a measurement of how realistic a human perceive a generated image and just maximize it. However, this is hard to quantify and it is still under research. Alternately there are different metrics such as visual fidelity of samples, log-likelihood, Parzen window estimates or Inception Score [35].

As it is shown in [36], the metrics to evaluate a generative model must be chosen in accordance with the objectives of the application. There is not an universal criterion that fits perfectly for all applications and goals of generative models. Furthermore, good performance with respect to one criterion does not necessarily imply good performance

in other metrics. For example, high log-likelihood does not guarantee realistic generated data [36]. Therefore, it is important to examine the goal when selecting the evaluation technique.

For this thesis, the goal is to assess generative models in terms of image synthesis and latent variable interpretability. As such, table 5.7 shows the metrics selected.

Quantitative Metrics	**Qualitative Metrics**
ELBO	Quality of generated images
Average log-likelihood	Latent space interpretability

Table 5.7. Metrics selected.

The ELBO function differs slightly between VAE (see Equation 3.8) and GMVAE (see Equation 4.3). It is the function maximized during training.

The average log-likelihood $E[\log p(\boldsymbol{x}|\boldsymbol{z})]$ measures the reconstruction capabilities of the model (it is not related to the generative capacity). It is the same for all models and has the following form.

$$E[\log p(\boldsymbol{x}|\boldsymbol{z})] \approx \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}\sum_{k=1}^{K}\log p(\boldsymbol{x}^{(i)}|\boldsymbol{z}_k^{(i)})$$

$$\log p(\boldsymbol{x}^{(i)}|\boldsymbol{z}_k^{(i)}) = -\frac{D_x}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(\boldsymbol{x}^{(i)} - \mu(\boldsymbol{z}_k^{(i)}))^2$$

Notice it is calculated sampling several $\boldsymbol{z}$ for each datum in the dataset to reach a better approximation.

The latent space interpretability will be appraised visually in terms of how well the multimodal nature of the data is represented in the latent space.

The quality of the generated images will be assessed through human visual interpretation. However, to simplify this task, only the models' configurations with smaller $KL\,(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z}))$ will be chosen in advance. Recall that a small $KL(*)$ means the distribution of the data in the latent space is close to the distribution from which generated data will be sampled.

5.9. Visualization of results

A very important aspect in deep learning is the visualization of the results. In this study the results are the generated / reconstructed images and the distribution of the latent variables. The visualization of the images is straightforward. However, the latent variables

sometimes are high dimensional so it is impossible to understand them in their raw dimension. Therefore, they are represented in 2D or 3D plots through a dimensionality reduction algorithm such as PCA or t-SNE.

6. RESULTS

This chapter analyzes the results obtained from the implementation of the previously described models. The source code is available at GitHub through the following URL: `https://github.com/psanch21/VAE-GMVAE`.

6.1. Initial considerations

This chapter contains several tables that show the value of the loss function (the negative *ELBO* function) at the end of the training process. These tables contain at least two columns called *ValidLoss* and *TrainingLoss* that refer to the negative *ELBO* function for the validation and train dataset respectively. These tables may contain other columns that represent the terms composing the *ELBO* function. Moreover, these tables contains a column called *ModelName* that holds information about the model configuration. For the case of VAE models (model1 or model2) this column has the following format *{model Type}_{dataset Name}_{epochs}_{sigma Recons}_{dimZ}_{dimHidden}_{num Layers}* and for the case of GMVAE models (model4_bias or model5) it has this format *{model Type}_{dataset Name}_{epochs}_{sigma Recons}_{dim Z}_{dim W}_{dim Hidden}_{num Layers}_{K clusters}*.

Another important consideration is that the range of the reconstruction loss and therefore the total loss depends on the chosen value for σ^2. So it is important to keep this in mind when comparing different configurations.

6.2. Experiment 1: VAE

Results for the different configurations of the VAE model show that it presents enormous difficulties to achieve good clustering and generation results at the same time. Tables 6.1 and 6.2 show the final loss of the configurations that resulted in the best clustering and generation capabilities respectively. The entire table with the results for all the configurations tested is included in Appendix A. Furthermore, Appendix B includes some examples of poorly generated samples.

ModelName	ValidLoss	TrainingLoss	ReconsLoss	KLloss	Recons/KL
model1_MNIST_300_001_10_128_4_exp1	603,49	566,01	529,24	36,77	14,39
model1_MNIST_300_001_10_128_3_exp1	619,90	584,35	548,43	35,92	15,27
model1_MNIST_300_001_10_64_4_exp1	670,69	647,08	611,34	35,74	17,11
model1_MNIST_300_0001_10_128_3_exp1	5890,58	5536,22	5481,34	54,89	99,87
model1_MNIST_300_0001_10_64_4_exp1	6418,49	6289,74	6235,66	54,09	115,28
model1_MNIST_300_0001_10_64_3_exp1	6522,76	6045,24	5990,76	54,47	109,97

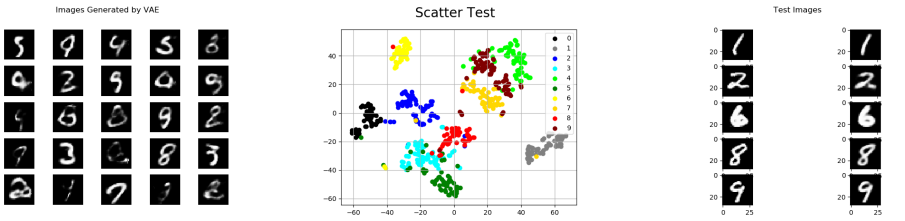
Table 6.1. Loss results of the best clustering configurations. Ordered by ValidLoss.

ModelName	ValidLoss	TrainingLoss	ReconsLoss	KLloss	Recons/KL
model1_MNIST_300_001_2_64_4_exp1	1489,76	1450,40	1438,28	12,12	118,66
model1_MNIST_300_001_2_64_2_exp1	1620,52	1479,28	1466,97	12,32	119,09
model1_MNIST_300_001_2_128_4_exp1	1434,47	1334,24	1321,13	13,12	100,73
model1_MNIST_300_0001_2_128_3_exp1	14412,44	14266,78	14229,07	37,71	377,37
model1_MNIST_300_0001_2_128_2_exp1	15847,81	14313,89	14269,89	44,00	324,29

Table 6.2. Loss results of the best generative configurations. Ordered by KLloss.

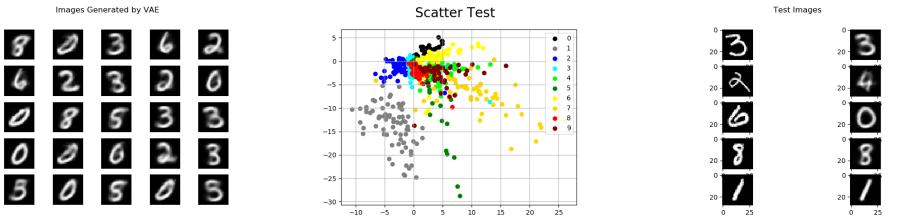
Analyzing the tables it is import to remark that the value of the KL and the dimension of the latent variable determine how the model behaves. Configurations that produce good clustering present larger dimension on the latent variable ($\dim z = 10$) and large KL (around $35 - 55$). On the other hand, configurations that obtain good generation are those with smaller dimension of the latent variable ($\dim z = 2$) and smaller KL (around $13 - 40$).

Figures 6.1 and 6.2 show examples of good clustering and good generation results respectively. In the first case, the multimodal nature is captured clearly but the generated samples are terrible, just a few of them can be considered realistic. The second case generates a lot of realistic samples (although a bit fuzzy). However, only the images representing the number 1 are visibly separated in the latent space, the others are mixed together.



(a) Generated images (b) Latent space (c) Reconstruction

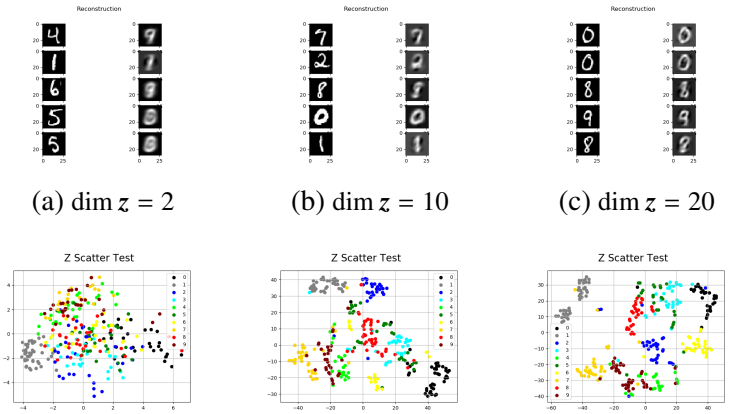
Fig. 6.1. Example of good clustering but poor generation result.



(a) Generated images (b) Latent space (c) Reconstruction

Fig. 6.2. Example of poor clustering but good generation result.

In any case, the results obtained with any configuration of the VAE outweigh the performance of the PCA algorithm in terms of image's reconstruction and latent space interpretability. As it is shown in Figure 6.3, the representation of the images in the latent space improves as the dimensionality increases, as with VAE. In relation to the reconstruction, it also improves with the dimensionality of the latent space, but the quality is clearly lower than with the VAE.



(a) dim $z = 2$ (b) dim $z = 10$ (c) dim $z = 20$

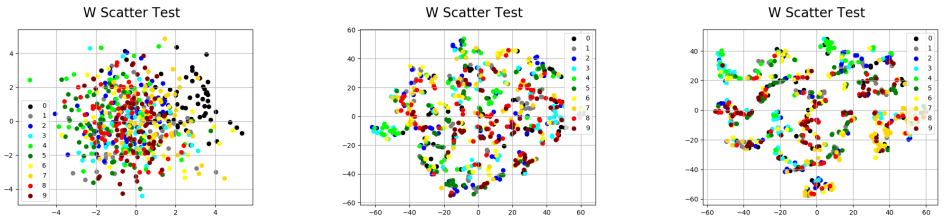
(d) dim $z = 2$ (e) dim $z = 10$ (f) dim $z = 20$

Fig. 6.3. Examples of PCA latent space representation and reconstruction varying dim z.

6.3. Experiment 2: Understanding the latent variables of GMVAE

Results show that the latent variable $\boldsymbol{w}$ holds information about variational features of the data while the latent variable z learns to separate the data in clusters, according to their class. Therefore, by fixing z and varying $\boldsymbol{w}$ is it possible to generate samples from the same class but slightly different.

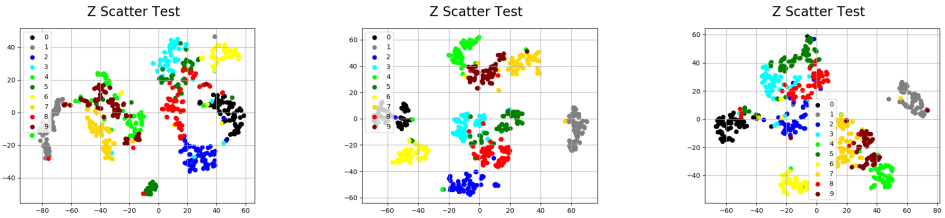
Figure 6.4 shows three scatter plots of the variable $\boldsymbol{w}$ with different dimensions. Representations in plot (a) are unmistakable distributed as Gaussian. Nonetheless, plots (b) and (c) show some kind of structure. However, it is attributed to the dimensionality reduction of the multidimensional Gaussian and therefore it is meaningless.



(a) dim $\boldsymbol{w}$ = 2 (b) dim $\boldsymbol{w}$ = 10 (c) dim $\boldsymbol{w}$ = 20

Fig. 6.4. Scatter plot of variable $\boldsymbol{w}$.

Figure 6.5 shows three scatter plots of the variable $\boldsymbol{z}$. It can be observed that a good clustering behaviour is obtained in all cases.



(a) dim $\boldsymbol{w}$ = 2 (b) dim $\boldsymbol{w}$ = 10 (c) dim $\boldsymbol{w}$ = 20

Fig. 6.5. Scatter plot of variable $\boldsymbol{z}$.

6.4. Experiment 3: Comparison between GMVAE and VAE

By comparing the loss values obtained in this experiment (which are shown in tables 6.3 and 6.4) with the values obtained in experiment 1, there is no significant improvements: neither *ValidLoss* nor the regularizer terms values are enhanced. Therefore, it may seem that the GMVAE has not proven better performance.

However, if the results are analyzed qualitatively (via visual appraisal) it can be noted significant performance improvements regarding interpretability of the latent space. Now, the variable $\boldsymbol{z}$ unambiguously captures the multimodal nature of the data and simultaneously the quality of the samples is quite good. So using the GMVAE is possible to accomplish both objectives, which was not possible in the standard VAE. Moreover, with the GMVAE it is possible to easily generate samples from a single class of the data in that each cluster is associated with a specific label. This is also a major improvement with respect to the standard VAE.

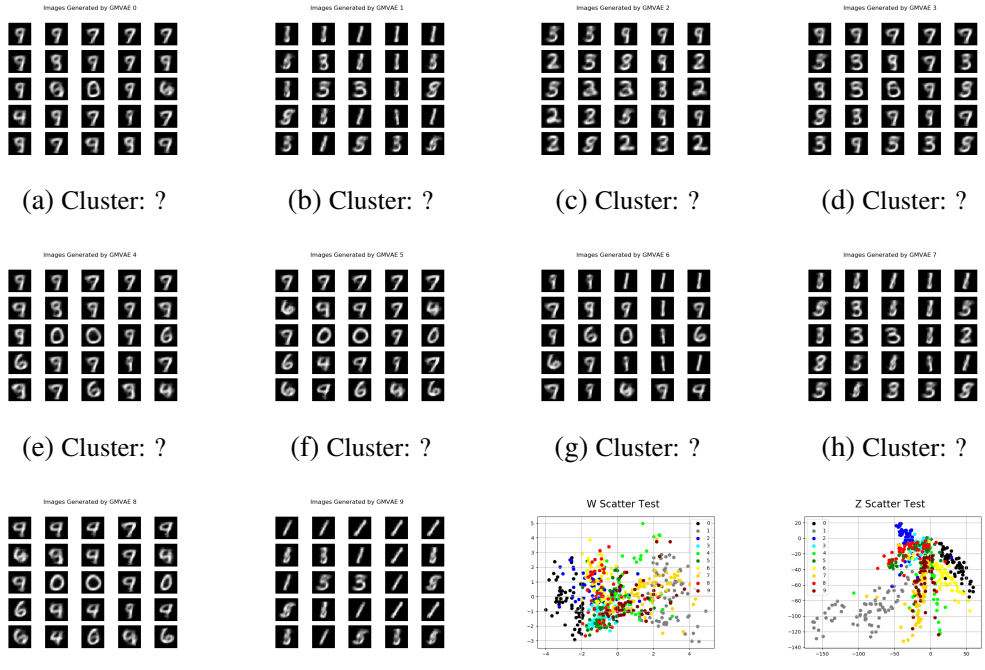
ModelName	ValidLoss	TrainLoss	ReconsLoss	CondPrior	WPrior	YPrior
model4_bias_MNIST_300_001_10_10_128_3_10_exp3	688,92	687,01	654,38	23,59	5,42	3,63
model4_bias_MNIST_300_001_10_2_128_3_10_exp3	690,86	640,15	608,10	25,38	3,39	3,28
model4_bias_MNIST_300_0001_10_10_128_3_10_exp3	6655,68	5891,97	5853,74	25,42	8,26	4,55
model4_bias_MNIST_300_001_10_10_128_4_10_exp3	692,20	710,49	676,98	25,87	4,08	3,56
model4_bias_MNIST_300_001_10_2_128_4_10_exp3	693,01	695,37	661,27	27,71	2,73	3,66
model4_bias_MNIST_300_0001_10_10_128_4_10_exp3	6793,94	6150,35	6108,40	30,12	7,32	4,50
model4_bias_MNIST_300_0001_10_2_128_3_10_exp3	6751,60	6109,63	6064,61	36,77	3,86	4,39
model4_bias_MNIST_300_0001_10_2_128_4_10_exp3	6765,19	5810,47	5762,56	38,80	4,64	4,47
model4_bias_MNIST_300_001_20_10_128_2_10_exp3	463,82	420,66	369,13	41,14	6,99	3,40
model4_bias_MNIST_300_001_20_2_128_2_10_exp3	470,04	474,70	421,08	47,63	2,78	3,21
model4_bias_MNIST_300_0001_20_10_128_4_10_exp3	3911,32	3916,91	3839,32	61,77	11,00	4,82
model4_bias_MNIST_300_0001_20_2_128_4_10_exp3	4037,11	4067,20	3972,26	83,07	7,21	4,66

Table 6.3. Best clustering configurations of VAE in GMVAE. Ordered by CondPrior.

ModelName	ValidLoss	TrainLoss	ReconsLoss	CondPrior	WPrior	YPrior
model4_bias_MNIST_300_0001_2_2_128_3_10_exp3	16218,51	15807,15	15795,70	5,85	2,66	2,94
model4_bias_MNIST_300_001_2_2_64_3_10_exp3	1646,86	1634,89	1623,81	6,11	2,36	2,60
model4_bias_MNIST_300_001_2_2_128_4_10_exp3	1625,47	1614,29	1602,08	6,27	2,74	3,20
model4_bias_MNIST_300_001_2_2_64_4_10_exp3	1682,63	1690,01	1677,47	6,60	2,97	2,96
model4_bias_MNIST_300_0001_2_2_64_4_10_exp3	16469,60	16438,52	16424,54	7,32	2,96	3,70
model4_bias_MNIST_300_0001_2_2_128_4_10_exp3	16065,48	15089,58	15073,15	9,20	3,49	3,75

Table 6.4. The best generative configurations of VAE in GMVAE. Ordered by CondPrior.

Figure 6.6 shows an example of a hyperparameters' configuration that led to good generation results in the VAE (see figure 6.2). It can be observed no substantial improvement as neither variable w nor z learn the multimodal nature of the data and the quality of the samples is similar.



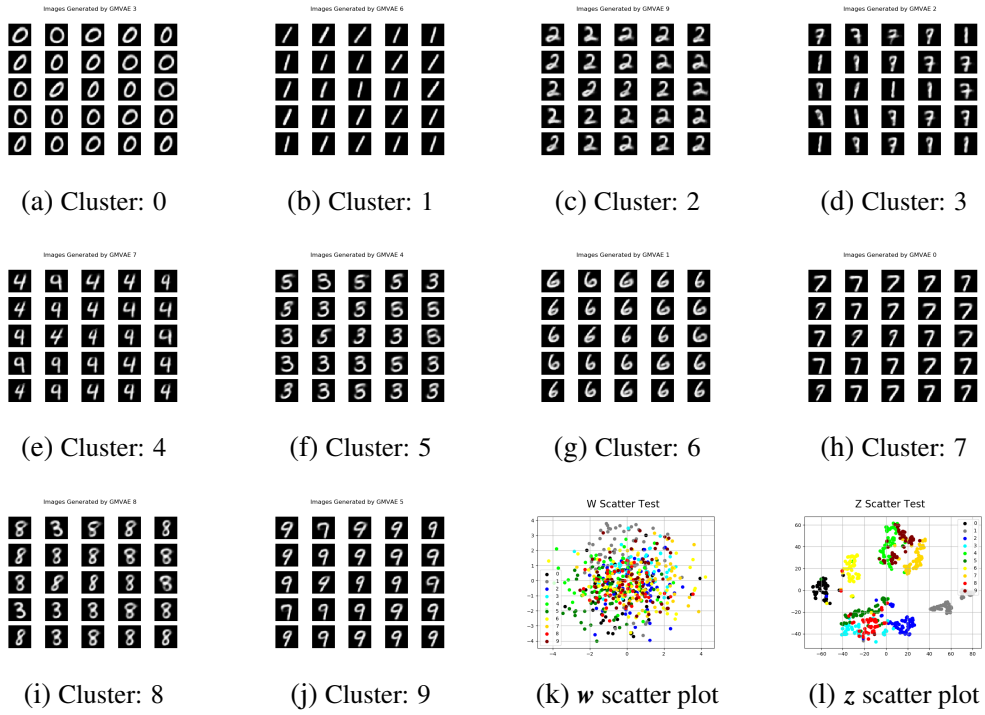
(a) Cluster: ? (b) Cluster: ? (c) Cluster: ? (d) Cluster: ?

(e) Cluster: ? (f) Cluster: ? (g) Cluster: ? (h) Cluster: ?

(i) Cluster: ? (j) Cluster: ? (k) w scatter plot (l) z scatter plot

Fig. 6.6. Example of bad clustering but good generation result.

Figure 6.7 shows an example of a hyperparameters' configuration that led to good clustering results but bad generation in the VAE (see figure 6.1). It can be observed using the GMVAE the quality of the samples generated is much better while keeping a good clustering in the latent space.



(a) Cluster: 0 (b) Cluster: 1 (c) Cluster: 2 (d) Cluster: 3

(e) Cluster: 4 (f) Cluster: 5 (g) Cluster: 6 (h) Cluster: 7

(i) Cluster: 8 (j) Cluster: 9 (k) w scatter plot (l) z scatter plot

Fig. 6.7. Example of good clustering and good generation result.

6.5. Experiment 4: Performance evaluation with corrupted input

Table 6.5 shows the loglikelihood without noise and with two different types of noise (uniform and dropout) for two configurations of the VAE, first two rows, and two configurations of the GMVAE, last two rows, that achieved good performance.

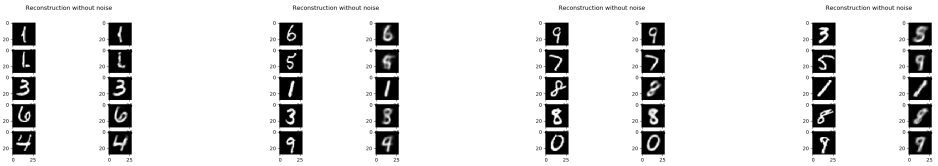
Before describing the results, it is important to notice the order of magnitude of the log-likelihood depends on the value of σ^2 as it appears in the denominator in the formula of the Gaussian log-likelihood.

Results demonstrate the VAE deteriorates much more than GMVAE when noise is introduced. The deterioration of the log-likelihood for both VAE's configurations is of 906.31 and 8136.39 units. On the contrary, the deterioration of the log-likelihood for the GMVAE is of 740.79 and 4201.7 units. So it is considerably less, which means the GMVAE is more resistant to corrupted inputs.

Id	ModelName	LogLikelihood	LogLikelihoodUniform	LogLikelihoodDropout
1	model1_MNIST_300_001_10_128_3_exp1	515,71	245,20	-390,60
2	model1_MNIST_300_0001_2_128_2_exp1	-14487,98	-19277,23	-22624,37
3	model4_bias_MNIST_300_001_10_2_128_3_10_exp3	438,90	309,08	-301,89
4	model4_bias_MNIST_300_0001_2_2_64_4_10_exp3	-14526,58	-14126,07	-18728,28

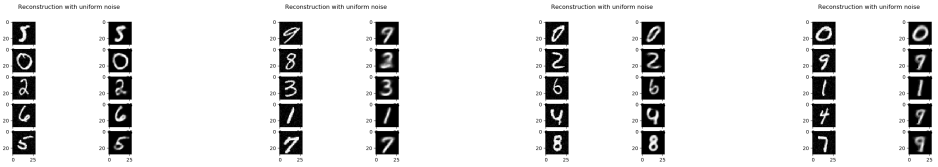
Table 6.5. Loglikelihood results.

Figures 6.8, 6.9 and 6.10 show examples of images inputted to the models (left column) and their reconstructions (right column) when different types of noise are present.



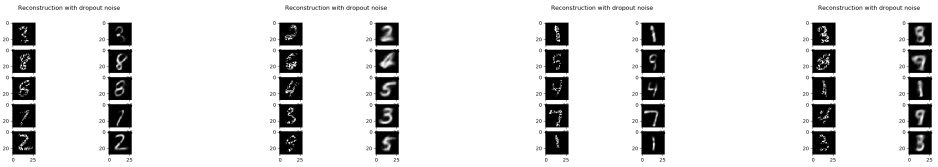
(a) Id: 1 (b) Id: 2 (c) Id: 3 (d) Id: 4

Fig. 6.8. Reconstruction without noise.



(a) Id: 1 (b) Id: 2 (c) Id: 3 (d) Id: 4

Fig. 6.9. Reconstruction with uniform noise.



(a) Id: 1 (b) Id: 2 (c) Id: 3 (d) Id: 4

Fig. 6.10. Reconstruction with dropout noise.

6.6. Experiment 5: Convolutional architecture

Table 6.6 and Table 6.7 show the values of the loss function have not significantly improved. They are smaller but remains in the same order as in experiments 3 and experiment 1 (see Tables 6.1, 6.2, 6.3 and 6.4).

ModelName	ValidLoss	TrainLoss	ReconsLoss	CondPrior	WPrior	YPrior
model5_MNIST_300_0001_10_2_64_3_10_exp5	4135,57	3306,13	3229,72	64,93	6,61	4,87

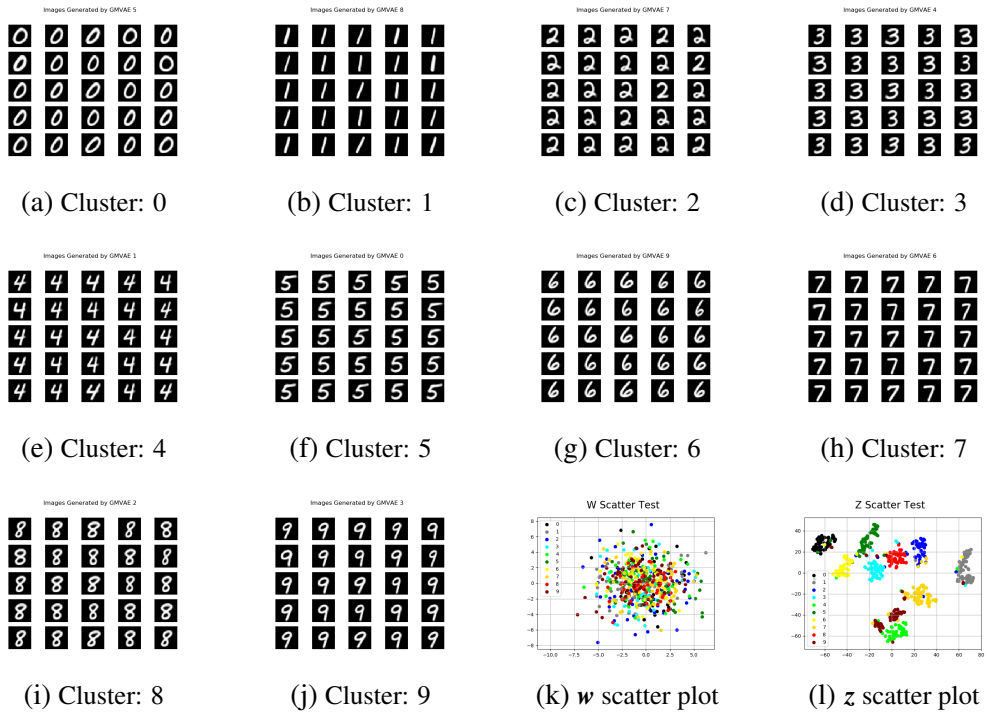
Table 6.6. GMVAE training result.

ModelName	ValidLoss	TrainLoss	ReconsLoss	KLloss
model2_MNIST_300_0001_10_64_3_exp5	3981,20	2731,63	2678,75	52,88
model2_MNIST_300_0001_2_128_2_exp5	13199,16	11867,55	11814,14	53,41

Table 6.7. VAE training result.

However it can be observed in Figure 6.11 and Figure 6.12 that excellent results are obtained for both GMVAE and VAE. For the GMVAE, the quality of the generated images improves dramatically and the data multimodality is perfectly represented in the latent space. This implies the generation is more accurate in the sense that each cluster generates samples only from a single class of the data. For the VAE, the results are improved compared to experiment 1. Nevertheless, the GMVAE clearly outweigh performance.

Another major improvement is that the number of parameters of the model is smaller so it is less prone to suffer from overfitting.

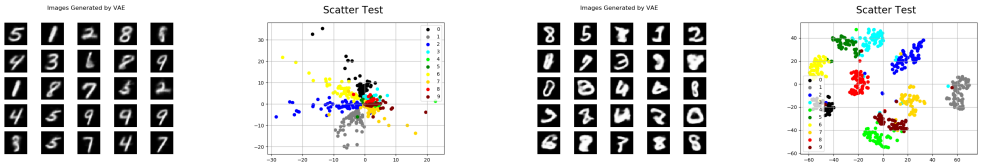


(a) Cluster: 0 (b) Cluster: 1 (c) Cluster: 2 (d) Cluster: 3

(e) Cluster: 4 (f) Cluster: 5 (g) Cluster: 6 (h) Cluster: 7

(i) Cluster: 8 (j) Cluster: 9 (k) $\boldsymbol{w}$ scatter plot (l) $\boldsymbol{z}$ scatter plot

Fig. 6.11. Example of results using GMVAE convolutional architecture on MNIST.

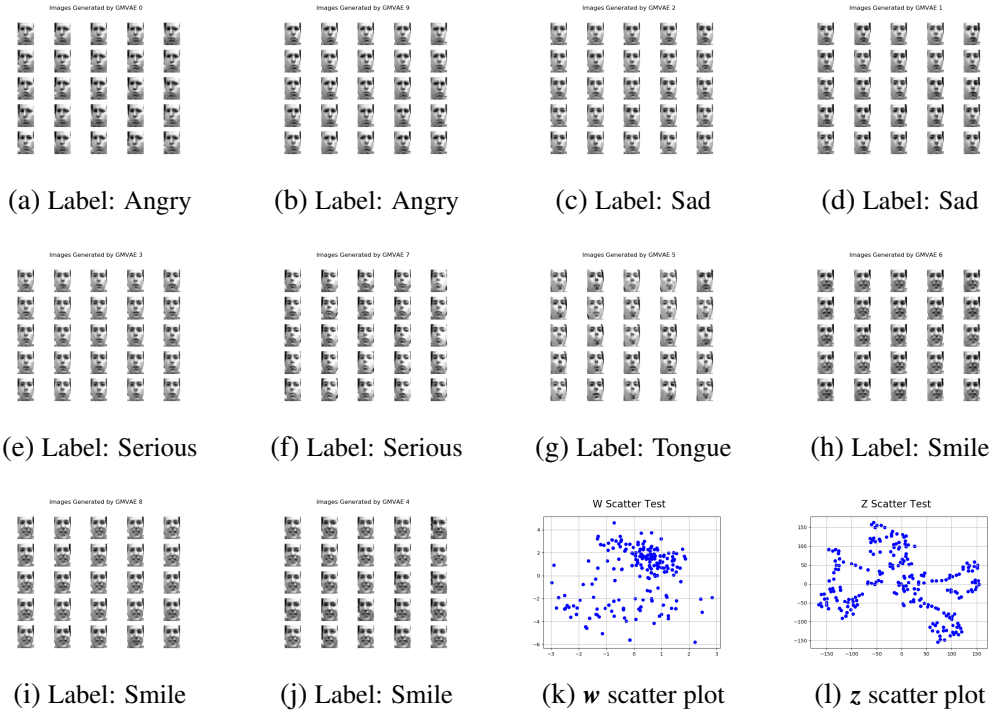


(a) Generated images (b) Latent space (c) Generated images (d) Latent space

Fig. 6.12. Example of results using VAE convolutional architecture on MNIST. Model (a) and (b) with dim $z = 2$ and model (c) and (d) with dim $z = 10$.

It has also been tested the GMVAE's convolutional architecture with the FREY dataset. Figure 6.13 shows that good results are obtained. It can be observed each cluster represents a well-defined expression and face orientation and therefore each cluster can be assigned to a specific label. Arbitrarily, five different classes have been defined: angry, sad, serious, tongue and smile. So now it is possible to obtain a labeled dataset.

It is important to mention that several clusters are assigned to the same label. There are two explanations for this. The first one is because training data present unbalanced samples from the different defined labels and the second one is that a label encapsulates a wider spectrum of variations. For example, there are many more faces smiling than sticking out the tongue so consequently three clusters are assigned to the label "smile" and one cluster is assigned to the label "tongue".



(a) Label: Angry (b) Label: Angry (c) Label: Sad (d) Label: Sad

(e) Label: Serious (f) Label: Serious (g) Label: Tongue (h) Label: Smile

(i) Label: Smile (j) Label: Smile (k) $\boldsymbol{w}$ scatter plot (l) $\boldsymbol{z}$ scatter plot

Fig. 6.13. Example of results using GMVAE's convolutional architecture on FREY dataset.

7. SOCIO-ECONOMIC ENVIRONMENT

This chapter presents the budget required to conduct a research project related to this thesis and possible practical applications of the VAE and GMVAE models alongside with the socio-economic impact they may generate.

7.1. Budget

A research project related to the study conducted in this thesis would require a budget to cover mainly human resources, computer power, conference costs and publication expenses. This section details the budget required to conduct a 2 years research project as part of the Signal Processing Group (known in Spanish as GTS) of the UC3M. The necessary funding will be € 50988.88. For the detailed budget see Table 7.1.

Regarding human resources costs, two people will be involved in the project: the principal investigator, which is already a UC3M professor, and a research student to be hired for the project. According to the official remuneration's tables of the UC3M, the monthly gross salary of a research student in the UC3M is € 1582.87 thus € 37988.88 in total.

In relation to the necessary equipment, the computer power costs are covered by the GTS. It has six servers with several Graphics Processing Units (GPUs) which are sufficient resources to conduct the experiments. The laptop for the student will have a cost of € 2000 and the cost of the personal computer for the principal investigator will be covered by the university. Software licenses are free of charge as the programming language and libraries needed are open source.

The part of the budget for conferences is estimated to be about € 3000 per year adding € 6000 for the whole project. It will cover transportation and accommodation costs and entry fees. Finally, according to the European Union, all results obtained in projects funded by public goods must be published. The expenses for publishing the results is estimated to be around € 5000 for the whole project.

Concept	Description	Cost €
Student's Expenses	Salary of the research student	37988.88
Student's PC	Laptop used to develop the implementation of the models and to connect to the server.	2000
Conferences' Expenses	Costs for traveling, accommodations and entry fees.	6000
Publications' Expenses	Costs for traveling, accommodations and entry fees.	5000
TOTAL		50988.88

Table 7.1. Detailed budget.

7.2. Practical Applications

The results obtain in this thesis can be extrapolated to more complicated problems. The family of variational autoencoders is suitable to be used in any situation where data is unlabeled and the objective is to generate new realistic samples, to understand the underlying the structure or to reconstruct data. Moreover, they can also be used to perform unsupervised clustering and obtain labeled samples that can be used to perform other tasks.

Table 7.2 shows some specific applications of generative models that contribute to accelerate processes and save time (market segmentation), improve the quality of the product (3D modeling) or provide with the required tools to develop other research projects (medical data generation). So generative models can contribute to save money and time, boost sales and encourage other researches.

Concept	Description
Image Denoising	Generative models can assist in reconstructing a corrupted image. For example, photographs taken under low light.
3D Modeling	Generative models can assist in the design of real world shapes for sandbox video games such as Minecraft. Generative models provide huge variety in the generated objects which can create a positive effect on the player.
Market segmentation	Generative models can be used to find a meaningful latent representation of users' data so that it is possible to identify different types of customer purchasing behaviour, create a labeled dataset that can be used to further analysis such as classification or regression.
Medical data generation	Medical data is usually scarce, hard to obtain and chaotic. Besides, it is extremely sensitive personal data in which privacy is really important. Generative models can generate new anonymous samples that can be used to perform further analysis.
Complex simulations	In fields such as astrophysics it is extremely expensive to conduct experiments to obtain data that can be analyzed. Given a dataset, generative models can generate more data and consequently save huge amount of money and time.

Table 7.2. Practical applications.

8. CONCLUSIONS

This chapter presents the conclusions of this thesis, revises the objectives and hypothesis stated at the beginning of the document and proposes future lines of research.

8.1. Conclusions

This thesis began as a research study whose goal was to fully understand one of the most widely used generative models family, the variational autoencoders, and compare the performance of the standard configuration with one of its variations. Analyzing the previous chapters, it can be safely said the objectives set at the beginning of this document have been more than fulfilled.

In a first step, the standard VAE was implemented smoothly, without any significant incident. Then, the implementation of the GMVAE was based on the objective function described in this thesis and the results obtained are quite successful. The visual quality of the generated samples and the latent representation is extraordinary, they have definitely improved compared to the standard VAE. It has been proven GMVAE outweigh the performance of standard VAE.

The realization of the experiments described previously in this work has helped to gain further understanding of how the models work. It has been observed the trade-off between good generative capabilities and good latent space representation of the VAE. Its simple definition of the latent space make it difficult to achieve both at the same time. Then it has been proven GMVAE eliminates this constrain successfully by defining a more expressive latent space. It has also been learnt about the importance of the dimensionality of the latent variables as it can determine the features of the resulting model. It has been observed that defining a high dimensional latent variable in the VAE results in a model that gives priority to reconstruction and good latent space representation over generation. This is likewise observed in the loss function in the form of high values for the KL divergence. It has also been assessed the GMVAE is more resilient to corrupted input as was expected. Finally, it has been appraised the importance of selecting the correct architecture: using CNNs the performance improves substantially and the number of parameters of the model decreases, which is desirable.

The main problem that has arose was the numerical instability found during training of the GMVAE. It was a time-consuming complication but it was also useful to learn how to debug neural networks and to learn how to search for alternative implementations, evaluate them and select the more suitable.

From a personal point of view, this thesis has been an introduction to the research world for me. I have acquired a solid knowledge about the mathematical and probabilistic

foundations of the state of the art generative models and I have learnt about how to implement them using one of the most famous frameworks for deep and machine learning. Furthermore, the source code is documented and is available in my personal GitHub, so hopefully it is useful for other deep learning enthusiasts. I am sure all the work and effort devoted to this thesis will be very helpful for future projects.

8.2. Future work

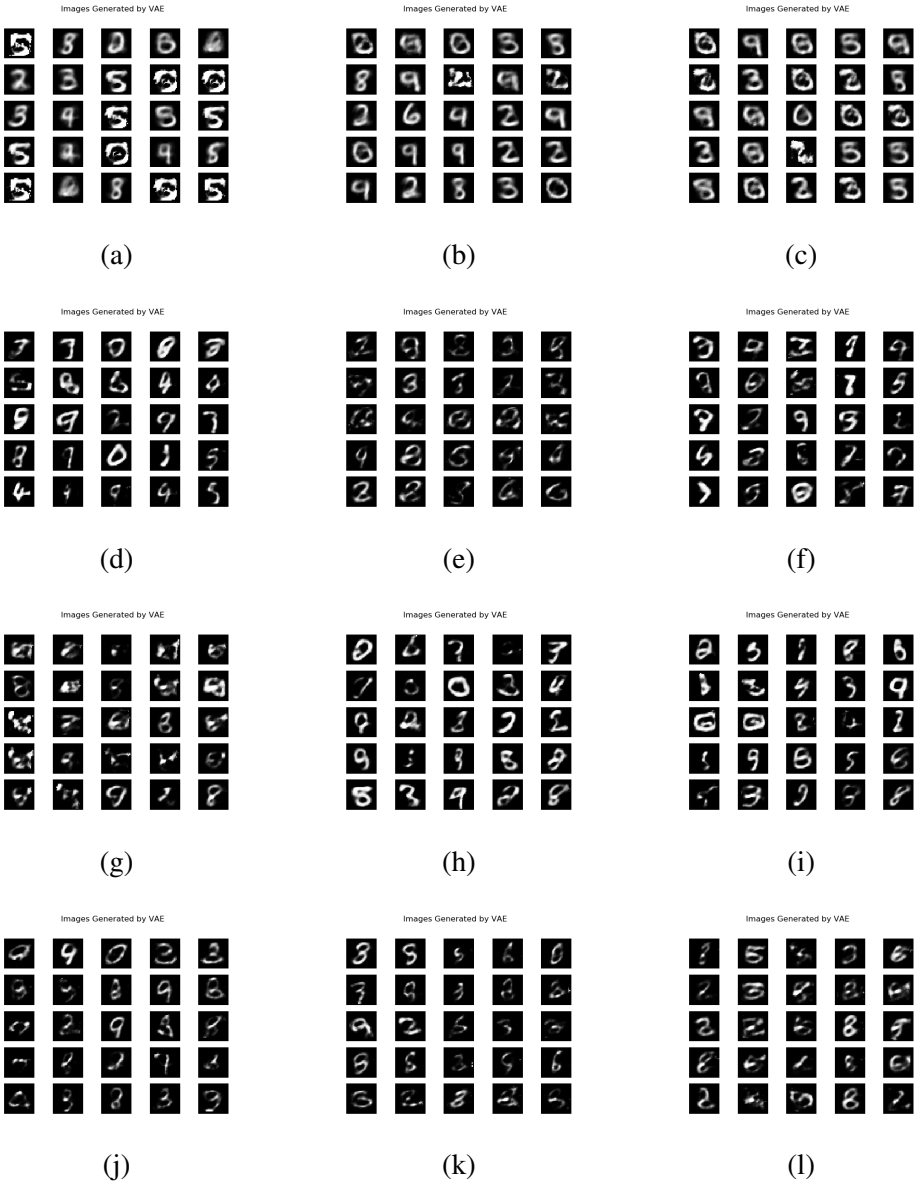
To improve the generative models presented in this document, in future work, more flexible assumptions on the nature of the distributions can be investigated. For example, not assuming that Σ_ϕ is a diagonal matrix. Finding more structured latent spaces could be an interesting line of research too. It could also be really useful to build a model for heterogeneous data in which it is possible to combine different continuous and discrete distributions and also add the temporal dimension.

A. VAE LOSS RESULTS

This appendix includes the table with the training results for the model configuration tested in experiment1.

ModelName	ValidLoss	TrainingLoss	ReconsLoss	KLloss	Recons/KL
model1_MNIST_300_001_50_128_3_exp1	265,71	267,10	164,07	103,03	1,59
model1_MNIST_300_001_50_128_4_exp1	268,07	268,79	170,04	98,75	1,72
model1_MNIST_300_001_50_128_2_exp1	269,86	246,59	147,48	99,11	1,49
model1_MNIST_300_001_50_64_2_exp1	327,14	331,34	230,79	100,55	2,30
model1_MNIST_300_001_50_64_3_exp1	337,39	320,27	224,21	96,07	2,33
model1_MNIST_300_001_50_64_4_exp1	353,51	341,57	253,55	88,02	2,88
model1_MNIST_300_001_20_128_3_exp1	371,13	319,51	260,83	58,68	4,44
model1_MNIST_300_001_20_128_2_exp1	388,32	341,05	282,35	58,70	4,81
model1_MNIST_300_001_20_128_4_exp1	401,09	423,40	361,34	62,06	5,82
model1_MNIST_300_001_20_64_4_exp1	445,76	437,52	375,96	61,56	6,11
model1_MNIST_300_001_20_64_3_exp1	449,42	414,88	353,97	60,91	5,81
model1_MNIST_300_001_20_64_2_exp1	458,40	458,97	400,83	58,15	6,89
model1_MNIST_300_001_10_128_4_exp1	603,49	566,01	529,24	36,77	14,39
model1_MNIST_300_001_10_128_3_exp1	619,90	584,35	548,43	35,92	15,27
model1_MNIST_300_001_10_128_2_exp1	634,17	569,60	532,79	36,81	14,47
model1_MNIST_300_001_10_64_4_exp1	670,69	647,08	611,34	35,74	17,11
model1_MNIST_300_001_10_64_3_exp1	710,51	598,71	563,77	34,94	16,13
model1_MNIST_300_001_10_64_2_exp1	732,48	773,94	738,89	35,05	21,08
model1_MNIST_300_001_2_128_4_exp1	1434,47	1334,24	1321,13	13,12	100,73
model1_MNIST_300_001_2_128_3_exp1	1463,60	1358,70	1342,70	16,00	83,92
model1_MNIST_300_001_2_64_4_exp1	1489,76	1450,40	1438,28	12,12	118,66
model1_MNIST_300_001_2_64_3_exp1	1527,01	1492,46	1479,38	13,09	113,05
model1_MNIST_300_0001_50_128_4_exp1	1549,46	1410,44	1238,47	171,97	7,20
model1_MNIST_300_0001_50_128_3_exp1	1556,36	1374,15	1203,34	170,80	7,05
model1_MNIST_300_001_2_128_2_exp1	1596,17	1285,37	1270,68	14,69	86,52
model1_MNIST_300_001_2_64_2_exp1	1620,52	1479,28	1466,97	12,32	119,09
model1_MNIST_300_0001_50_128_2_exp1	1635,30	1491,24	1317,90	173,34	7,60
model1_MNIST_300_0001_50_64_4_exp1	2115,31	2103,00	1930,18	172,82	11,17
model1_MNIST_300_0001_50_64_3_exp1	2142,58	1794,30	1626,81	167,49	9,71
model1_MNIST_300_0001_50_64_2_exp1	2150,82	2162,38	1994,37	168,01	11,87
model1_MNIST_300_0001_20_128_4_exp1	3106,94	2945,58	2853,25	92,33	30,90
model1_MNIST_300_0001_20_128_3_exp1	3169,56	2729,46	2637,39	92,07	28,65
model1_MNIST_300_0001_20_128_2_exp1	3472,80	3450,75	3354,46	96,29	34,84
model1_MNIST_300_0001_20_64_4_exp1	3885,45	3939,61	3846,32	93,28	41,23
model1_MNIST_300_0001_20_64_3_exp1	3969,36	3675,23	3584,08	91,15	39,32
model1_MNIST_300_0001_20_64_2_exp1	4068,23	4047,44	3955,24	92,20	42,90
model1_MNIST_300_0001_10_128_4_exp1	5781,98	4999,60	4946,13	53,47	92,51
model1_MNIST_300_0001_10_128_3_exp1	5890,58	5536,22	5481,34	54,89	99,87
model1_MNIST_300_0001_10_128_2_exp1	6081,59	5401,70	5348,67	53,03	100,86
model1_MNIST_300_0001_10_64_4_exp1	6418,49	6289,74	6235,66	54,09	115,28
model1_MNIST_300_0001_10_64_3_exp1	6522,76	6045,24	5990,76	54,47	109,97
model1_MNIST_300_0001_10_64_2_exp1	6796,34	7501,34	7446,23	55,10	135,14
model1_MNIST_300_0001_2_128_4_exp1	13879,88	13084,50	13023,77	60,73	214,44
model1_MNIST_300_0001_2_128_3_exp1	14412,44	14266,78	14229,07	37,71	377,37
model1_MNIST_300_0001_2_64_4_exp1	14641,84	15802,50	15763,55	38,95	404,68
model1_MNIST_300_0001_2_64_3_exp1	14887,04	13993,33	13956,44	36,89	378,28
model1_MNIST_300_0001_2_128_2_exp1	15847,81	14313,89	14269,89	44,00	324,29
model1_MNIST_300_0001_2_64_2_exp1	16114,93	14350,68	14296,52	54,16	263,97

B. GENERATED SAMPLES



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Fig. B.1. Low quality generated samples with different VAE configurations.

C. KL DIVERGENCE OF GMM

Recall that the Kullback-Leibler divergence between two distributions is defined as follows:

$$KL(q||p) = \int p(x) log \frac{p(x)}{q(x)} dx = \int p(x) log p(x) dx - \int p(x) log q(x) dx = I_1 + I_2$$

Considering the case for two univariate variables, let $q(x)$ be a Gaussian Mixture Model with c components with different means and variances, each of them with probability Π_i ($\sum_{i=1}^{c} \Pi_i = 1$). On the other hand, $p(x)$ is a Gaussian variable with a certain mean and variance:

$$q(x) = \sum_{i=1}^{c} \Pi_i N(\mu_i, \sigma_i)$$

$$p(x) = N(\mu_1, \sigma_1)$$

Then, the integral I_1 is easy to solve and it has a closed form:

$$I_1 = \int \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} log(\frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}) dx = \int \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} (-\frac{1}{2} log(2\pi\sigma_1^2) - \frac{(x-\mu_1)^2}{2\sigma_1^2}) dx$$

$$\downarrow$$

$$I_1 = -\frac{1}{2}(1 + log(2\pi\sigma_1^2))$$

However, the integral I_2 does not have a closed solution, it contains a summation inside a logarithm:

$$I_2 = \int p(x) log(\sum_{i=1}^{c} \Pi_i N(\mu_i, \sigma_i)) dx = \int p(x) log(\sum_{i=1}^{c} \Pi_i \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}) dx$$

As a result, the KL divergence when a GMM distribution is involved is not analytically tractable [37].

D. GMVAE OBJECTIVE FUNCTION DERIVATIONS.

D.1. Conditional prior term

$$E_q\left[\log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y)}\right] = \int_z\int_w\int_y q_{\beta_y,\phi_z,\phi_w}(\boldsymbol{z},\boldsymbol{w},y|\boldsymbol{x}) \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y)} dy d\boldsymbol{w} d\boldsymbol{z} =$$

$$\int_z\int_w\int_y q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x}) q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x}) p_\beta(y|\boldsymbol{w},\boldsymbol{z}) \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y)} dy d\boldsymbol{w} d\boldsymbol{z} =$$

$$\int_w q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x}) \int_z q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x}) \left(\sum_{k=1}^K p_\beta(y_k=1|\boldsymbol{w},\boldsymbol{z}) \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right) d\boldsymbol{z} d\boldsymbol{w} \rightarrow$$

$$E_q\left[\log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y)}\right] = E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x}) q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^K \tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right]$$

This term will be estimated using Monte Carlo estimator:

$$E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x}) q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^K \tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right] \approx \frac{1}{N}\frac{1}{M}\sum_{n=1}^N\sum_{m=1}^M\sum_{k=1}^K \tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}_m|\boldsymbol{x})}{p_\beta(\boldsymbol{z}_m|\boldsymbol{w}_n,y_k=1)}$$

$$q_{\phi_z}(\boldsymbol{z}_m|\boldsymbol{x}) \rightarrow \boldsymbol{z}_m = \mu_{\phi_z}(\boldsymbol{x}) + \sigma^2_{\phi_z}(\boldsymbol{x}) * \epsilon$$

$$p_\beta(\boldsymbol{z}_m|\boldsymbol{w}_n,y_k=1) \rightarrow \boldsymbol{z}_m = \mu_\beta(\boldsymbol{w})^{y_k} + \sigma^2_\beta(\boldsymbol{w})^{y_k} * \epsilon$$

To simplify the calculations only 1 sample will be considered:

$$E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x}) q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^K \tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right] \approx \sum_{k=1}^K \tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}_m|\boldsymbol{x})}{p_\beta(\boldsymbol{z}_m|\boldsymbol{w}_n,y_k=1)} =$$

$$\sum_{k=1}^K \tilde{\pi}_k \Big(\log q_{\phi_z}(\boldsymbol{z}_m|\boldsymbol{x}) - \log p_\beta(\boldsymbol{z}_m|\boldsymbol{w}_n,y_k=1)\Big) = \log q_{\phi_z}(\boldsymbol{z}_m|\boldsymbol{x}) \sum_{k=1}^K \tilde{\pi}_k - \sum_{k=1}^K \tilde{\pi}_k \log p_\beta(\boldsymbol{z}_m|\boldsymbol{w}_n,y_k=1) =$$

It is known that $\sum_{k=1}^K \tilde{\pi}_k = 1$ then:

$$E_{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x}) q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^K \tilde{\pi}_k \log \frac{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{w},y_k=1)}\right] = \log q_{\phi_z}(\boldsymbol{z}_m|\boldsymbol{x}) - \sum_{k=1}^K \tilde{\pi}_k \log p_\beta(\boldsymbol{z}_m|\boldsymbol{w}_n,y_k=1) \tag{D.1}$$

Each of the terms in equation D.1 can be further developed:

$$q_{\phi_z}(\boldsymbol{z}_m|\boldsymbol{x}) = N(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x})) \rightarrow \log q_{\phi_z} = cte - \frac{1}{2}\log\det\sigma^2(\boldsymbol{x}) - \frac{1}{2\sigma^2(\boldsymbol{x})}(z_m - \mu(\boldsymbol{x}))^2$$

$$p_\beta(\boldsymbol{z}_m|\boldsymbol{w}_n, y_k = 1) = N(\mu(\boldsymbol{w})^{y_k}, \sigma^2(\boldsymbol{w})^{y_k}) \rightarrow \log p_\beta = cte - \frac{1}{2}\log\det\sigma^2(\boldsymbol{w})^{y_k} - \frac{1}{2\sigma^2(\boldsymbol{w})^{y_k}}(z_m - \mu(\boldsymbol{w})^{y_k})^2$$

It is possible to further simplify the sumatory:

$$\sum_{k=1}^{K}\tilde{\pi_k}\log p_\beta(\boldsymbol{z}_m|\boldsymbol{w}_n, y_k = 1) = \sum_{k=1}^{K}\tilde{\pi_k}cte - \tilde{\pi_k}\frac{1}{2}\log\det\sigma^2(\boldsymbol{w})^{y_k} - \tilde{\pi_k}\frac{1}{2\sigma^2(\boldsymbol{w})^{y_k}}(z_m - \mu(\boldsymbol{w})^{y_k})^2 =$$

$$cte - \frac{1}{2}\sum_{k=1}^{K}\tilde{\pi_k}\log\det\sigma^2(\boldsymbol{w})^{y_k} - \frac{1}{2}\sum_{k=1}^{K}\tilde{\pi_k}\frac{1}{\sigma^2(\boldsymbol{w})^{y_k}}(z_m - \mu(\boldsymbol{w})^{y_k})^2 =$$

D.2. W-prior term

$$E_q\left[\log\frac{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}{p(\boldsymbol{w})}\right] = \int_z\int_w\int_y q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})p_\beta(y|\boldsymbol{w},\boldsymbol{z})\log\frac{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}{p(\boldsymbol{w})}dyd\boldsymbol{w}d\boldsymbol{z} =$$

$$\int_w q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})\log\frac{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}{p(\boldsymbol{w})}\left(\int_z q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})\left(\sum_{k=1}^{K}p_\beta(y_k = 1|\boldsymbol{w},\boldsymbol{z})\right)d\boldsymbol{z}\right)d\boldsymbol{w} =$$

As $\sum_{k=1}^{K}p_\beta(y_k = 1|\boldsymbol{w},\boldsymbol{z}) = 1$ and $\int_z q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{z} = 1$ the equation obtained is:

$$E_q\left[\log\frac{q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}{p(\boldsymbol{w})}\right] = KL\left(q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})||p(\boldsymbol{w})\right)$$

D.3. Y-prior term

$$E_q\left[\log\frac{p_\beta(y|\boldsymbol{w},\boldsymbol{z})}{p(y)}\right] = \int_z\int_w\int_y q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})p_\beta(y|\boldsymbol{w},\boldsymbol{z})\log\frac{p_\beta(y|\boldsymbol{w},\boldsymbol{z})}{p(y)}d\boldsymbol{w}d\boldsymbol{z} =$$

$$\int_z\int_w q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})\left(\sum_{k=1}^{K}p_\beta(y_k = 1|\boldsymbol{w},\boldsymbol{z})\log\frac{p_\beta(y_k = 1|\boldsymbol{w},\boldsymbol{z})}{p(y_k = 1)}\right)d\boldsymbol{w}d\boldsymbol{z} =$$

]

$$\int_z\int_w q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right)d\boldsymbol{w}d\boldsymbol{z} =$$

$$E_q\left[\log \frac{p_\beta(y|\boldsymbol{w},\boldsymbol{z})}{p(y)}\right] = E_{q_{\phi_z}(\boldsymbol{z}|\boldsymbol{x})q_{\phi_w}(\boldsymbol{w}|\boldsymbol{x})}\left[KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right)\right]$$

$$KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right) = \sum_{i=1}^{K} p(y_i=1)\log\frac{p(y_i=1)}{p_\beta(y_i=1|\boldsymbol{w},\boldsymbol{z})}$$

Recalling that $p(y_i=1)=\frac{1}{K}$

$$KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right) = \frac{1}{K}\sum_{i=1}^{K}\log\frac{1}{K} - \log p_\beta(y_i=1|\boldsymbol{w},\boldsymbol{z})$$

Finally

$$KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right) = -\frac{1}{K}\sum_{i=1}^{K}\log K + \log p_\beta(y_i=1|\boldsymbol{w},\boldsymbol{z})$$

$$KL\left(p_\beta(y|\boldsymbol{w},\boldsymbol{z})||p(y)\right) = -\log K - \frac{1}{K}\sum_{i=1}^{K}\log p_\beta(y_i=1|\boldsymbol{w},\boldsymbol{z})$$

BIBLIOGRAPHY

[1] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," in, J. A. Anderson and E. Rosenfeld, Eds., Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104451.

[3] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012. DOI: 10.1109/MSP.2012.2205597.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *ArXiv e-prints*, Jun. 2015. arXiv: 1506.02640 [cs.CV].

[5] A. Graves, "Generating Sequences With Recurrent Neural Networks," *ArXiv e-prints*, Aug. 2013. arXiv: 1308.0850.

[6] N. Dilokthanakul *et al.*, "Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders," *ArXiv e-prints*, Nov. 2016. arXiv: 1611.02648 [cs.LG].

[7] C. J. B. Yann LeCun Corinna Cortes. (2000). The mnist database of handwritten digits, [Online]. Available: http://yann.lecun.com/exdb/mnist/ (visited on 01/26/2018).

[8] S. Roweis. (2000). Frey faces dataset, [Online]. Available: https://cs.nyu.edu/~roweis/data.html (visited on 04/08/2018).

[9] European Union, "Regulation 2016/679 of the European parliament and the Council of the European Union," *Official Journal of the European Communities*, vol. 2014, no. April, pp. 1–88, 2016. DOI: http://eur-lex.europa.eu/pri/en/oj/dat/2003/l_285/l_28520031101en00330037.pdf. arXiv: arXiv:1011.1669v3.

[10] Jefatura del Estado, "Real Decreto legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia.," *Boletín oficial del estado*, pp. 14 369–14 396, 1996.

[11] I. Goodfellow, "NIPS 2016 Tutorial: Generative Adversarial Networks," *ArXiv e-prints*, Dec. 2017. arXiv: 1701.00160 [cs.LG].

[12] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," *ArXiv e-prints*, Jan. 2016. arXiv: 1601.06759 [cs.CV].

[13] L. Dinh, D. Krueger, and Y. Bengio, "NICE: Non-linear Independent Components Estimation," *ArXiv e-prints*, Oct. 2014. arXiv: 1410.8516 [cs.LG].

[14] S. Mohamed and B. Lakshminarayanan, "Learning in Implicit Generative Models," *ArXiv e-prints*, Oct. 2016. arXiv: 1610.03483 [stat.ML].

[15] G. Alain *et al.*, "GSNs : Generative Stochastic Networks," *ArXiv e-prints*, Mar. 2015. arXiv: 1503.05571 [cs.LG].

[16] I. J. Goodfellow *et al.*, "Generative Adversarial Networks," *ArXiv e-prints*, Jun. 2014. arXiv: 1406.2661 [stat.ML].

[17] J. Shlens, "A tutorial on principal component analysis," *CoRR*, vol. abs/1404.1100, 2014. arXiv: 1404.1100. [Online]. Available: http://arxiv.org/abs/1404.1100.

[18] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ch. 12, pp. 570–587.

[19] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. [Online]. Available: https://doi.org/10.1080/14786440109462720.

[20] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999. [Online]. Available: http://www.jstor.org/stable/2680726.

[21] S. T. Roweis, "Em algorithms for pca and spca," in *Advances in neural information processing systems*, 1998, pp. 626–632.

[22] S. Roweis, "EM Algorithms for PCA and SPCA," *Computing*, vol. 10, no. 13, pp. 626–632, 1997. DOI: 10.1021/ja100409b. arXiv: arXiv:1011.1669v3. [Online]. Available: https://cs.nyu.edu/{~}roweis/papers/empca.pdf.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. DOI: 10.1109/5.726791.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105, 2012. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *ArXiv e-prints*, Dec. 2015. arXiv: 1512.03385 [cs.CV].

[26] (). Gnu image manipulation program, [Online]. Available: https://docs.gimp.org/en/plug-in-convmatrix.html (visited on 05/28/2018).

[27] A. K. Justin Johnson. (2009). Cs231n: Convolutional neural networks for visual recognition, [Online]. Available: http://cs231n.github.io/convolutional-networks/#conv (visited on 04/20/2018).

[28] D. P Kingma and M. Welling, "Auto-Encoding Variational Bayes," *ArXiv e-prints*, Dec. 2013. arXiv: 1312.6114 [stat.ML].

[29] M. Shiffman. (2016). Under the hood of the variational autoencoder (in prose and code), [Online]. Available: http://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html (visited on 03/10/2018).

[30] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv e-prints*, Sep. 2016. arXiv: 1609.04747 [cs.LG].

[31] C. Doersch, "Tutorial on Variational Autoencoders," *ArXiv e-prints*, Jun. 2016. arXiv: 1606.05908 [stat.ML].

[32] J. Wang and J. X. Wang, "Properties of the Random Variable in Normal Distribution," *Nonlinear Analysis and Differential Equations*, vol. 5, no. 5, 2017. DOI: 10.12988/nade.2017.756. [Online]. Available: http://www.m-hikari.com/nade/nade2017/5-8-2017/p/wangNADE5-8-2017.pdf.

[33] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," *CoRR*, vol. abs/1406.5298, 2014. arXiv: 1406.5298. [Online]. Available: http://arxiv.org/abs/1406.5298.

[34] R. Shu. (2016). Gaussian mixture vae: Lessons in variational inference, generative models, and deep nets, [Online]. Available: http://ruishu.io/2016/12/25/gmvae/ (visited on 03/22/2018).

[35] T. Salimans *et al.*, "Improved Techniques for Training GANs," *ArXiv e-prints*, Jun. 2016. arXiv: 1606.03498 [cs.LG].

[36] L. Theis, A. van den Oord, and M. Bethge, "A note on the evaluation of generative models," *ArXiv e-prints*, Nov. 2015. arXiv: 1511.01844 [stat.ML].

[37] J. R. Hershey and P. A. Olsen, "Approximating the kullback leibler divergence between gaussian mixture models," vol. 4, Apr. 2007. DOI: 10.1109/ICASSP.2007.366913.