Deep Clustering Based on Implicit Likelihood Maximization

A Thesis

submitted to the designated
by the Assembly
of the Department of Computer Science and Engineering
Examination Committee

by

Georgios Vardakas

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER
SYSTEMS ENGINEERING

WITH SPECIALIZATION
IN DATA SCIENCE AND ENGINEERING

University of Ioannina
School of Engineering
Ioannina October 2021

Examining Committee:

- **Aristidis Likas**, Professor, Department of Computer Science & Engineering, University of Ioannina (Supervisor)
- **Konstantinos Blekas**, Professor, Department of Computer Science & Engineering, University of Ioannina
- **Christophoros Nikou**, Professor, Department of Computer Science & Engineering, University of Ioannina

Dedication

Dedicated to my family for its unconditional support throughout all the years of my studies.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Aristeidis Likas, for the valuable guidance, patience, and trust he showed me through all our collaboration. He was more than eager to provide me with his knowledge on the subject while making helpful suggestions all over this work. His guidance, support, and expertise have been only some of the ways through which he has shaped my first steps in research.

I would also like to thank my old friend and fellow student Ilias Kleftakis for his valuable help in our undergraduate studies and his assistance in setting the machine in which the experiments took place. Furthermore, I would like to thank my close friend and fellow student Spyros Tzimas for our discussions and the quality cooperation in our undergraduate studies. Finally, I want to thank my old friend Grigoris Papigiotis for his unconditional support and scientific discussions about Machine Learning and its applications in Astrophysics.

I would also like to thank my family for their unconditional support and love throughout the years. My mother, Athena, brother and sister Konstantinos and Eleni-Anastasia, who always supported me, and my father, Thomas, who taught me to demand more of myself. Moreover, I wish to thank my close friends Giannis, Sokratis, Sotiris, Theodore, and Christodoulos for their support and understanding.

Last but not least, I owe a special thanks to Evgenia Lampropoulou. She showed admirable care, support, and patience throughout all the undergraduate and postgraduate years of my studies and was enthusiastic about discussing numerous parts of it.

Abstract

Georgios Vardakas, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, October 2021.
Deep Clustering Based on Implicit Likelihood Maximization.
Advisor: Aristidis Likas, Professor.

Deep Learning is a type of machine learning and artificial intelligence that imitates how the human brain can learn. It is one of the essential elements of data science, which includes statistics and predictive modeling. Although DL started mainly for supervised tasks, lately, it has found success in several unsupervised learning fields, like clustering, dimensionality reduction, etc. Clustering belongs to unsupervised machine learning and is defined as a process of assigning objects to groups so that the data share common characteristics. Therefore, the main goal of clustering is for objects belonging to the same group to be similar (or related) to each other and differ (or not be related) to objects in different groups. This way, clustering explores the data and aims to find (hidden) structures in them. At the same time, clustering is one of the most challenging problems in the field of machine learning.

In this master's thesis, we study Deep Clustering methods. Deep clustering is a new promising area of clustering algorithms that emerged in recent years. The main goal of Deep Clustering is to create clustering algorithms merged with Deep Learning methods to exploit their representational power. Therefore, in this thesis, we will clearly describe the new machine learning area of Deep Clustering and why it is considered promising. Afterward, we will present two Deep Clustering algorithms that were studied. The first Deep Clustering algorithm that we will discuss is the ClusterGan, which makes use of a modified Generative Adversarial Networks' architecture in order to cluster the data in latent space $\mathcal{Z}$. The second Deep Clustering method that we will present is our contribution, and it is based on a generative Deep Neural

Network model that is trained by Implicit Likelihood Maximization (IMLE). IMLE provides an effective way of maximizing the likelihood of the model indirectly. The Deep Clustering methodology that is based on IMLE also clusters the data in the latent space. Finally, we will analyze the experiments that took place and present the experimental results.

Keywords: machine learning, deep learning, deep neural networks, generative data models, implicit models, implicit maximum likelihood estimation, IMLE, generative adversarial network, GANs, clustering, deep clustering, ClusterGan.

Εκτεταμενη Περιληψη

Γεώργιος Βαρδάκας, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, Οκτώβριος 2021.
Βαθιά Ομαδοποίηση Βασισμένη στην Έμμεσης Μεγιστοποίησης της Πιθανοφάνειας.
Επιβλέπων: Αριστείδης Λύκας, Καθηγητής.

Η βαθιά μάθηση είναι ο τομέας της μηχανικής μάθησης που βασίζεται στην εκπαίδευση νευρωνικών δικτύων μεγάλου βάθους (deep neural networks). Παρά το γεγονός ότι η βαθιά μάθηση εφαρμόζεται κυρίως σε προβλήματα μάθησης με επίβλεψη, πρόσφατα αρχίσει να χρησιμοποιείται για διάφορα προβλήματα χωρίς επίβλεψη, όπως είναι η ομαδοποίηση, η μείωση διάστασης κ.λπ. Η ομαδοποίηση ανήκει στην κατηγορία προβλημάτων μηχανικής μάθησης χωρίς επίβλεψη, και ορίζεται ως την διαδικασία ανάθεσης αντικειμένων (δεδομένων) σε ομάδες, ώστε αυτά να έχουν κοινά χαρακτηριστικά. Ο κύριος στόχος της ομαδοποίησης είναι τα αντικείμενα που ανήκουν στην ίδια ομάδα να είναι περισσότερο όμοια μεταξύ τους και να διαφέρουν με αντικείμενα άλλων ομάδων.

Στην παρούσα μεταπτυχιακή διπλωματική εργασία, μελετάμε μεθόδους βαθιάς ομαδοποίησης. Κύριος στόχος της βαθιάς ομαδοποίησης είναι η κατασκευή μεθόδων ομαδοποίησης, οι οποίες χρησιμοποιούν βαθιά μάθηση έτσι ώστε να εκμεταλλευτούν την ικανότητα που παρουσιάζουν τα βαθιά νευρωνικά δίκτυα στο να μετασχηματίζουν αποδοτικά τα δεδομένα σε μη γραμμικούς χώρους. Θα παρουσιάσουμε τους δύο βασικούς αλγορίθμους που μελετήθηκαν. Ο πρώτος αλγόριθμος βαθιάς ομαδοποίησης που θα παρουσιάσουμε είναι το ClusterGan, το οποίο κάνει χρήση μίας τροποποιημένης αρχιτεκτονικής παραγωγικών δικτύων ανταγωνιστικής μάθησης (GANs) με στόχο την ομαδοποίηση των δεδομένων. Η δεύτερη μεθοδολογία που θα παρουσιαστεί αποτελεί δική μας συνεισφορά, και βασίζεται σε ένα παραγωγικό νευρωνικό δίκτυο το οποίο εκπαιδεύεται μέσω έμμεσης μεγιστοποίησης της πιθανοφάνειας (IMLE). Αξίζει να σημειωθεί ότι η έμμεση μεγιστοποίηση της πιθανοφάνειας είναι μία αξιόλογη μέθοδος για την εκπαίδευση παραγωγικών δικτύων στην οποία θα επικεντρωθούμε και θα μελετήσουμε στην παρούσα εργασία. Η μέθοδος βαθιάς ομαδοποίησης η οποία βασίζεται στην έμμεση μεγιστοποίηση της πιθανοφάνειας έχει την δυνατότητα να ομαδοποιεί τα δεδομένα στο επίπεδο εισόδου του μοντέλου. Τέλος, θα παρουσιάσουμε τα πειράματα που που πραγματοποιήθηκαν και θα αναλύσουμε τα πειραματικά αποτελέσματα.

Λέξεις Κλειδιά: μηχανική μάθηση, βαθιά μάθηση, βαθιά νευρωνικά δίκτυα, παραγωγικά μοντέλα δεδομένων, έμμεσα μοντέλα, έμμεση μεγιστοποίηση της πιθανοφάνειας, παραγωγικά ανταγωνιστικά δίκτυα, ομαδοποίηση, βαθιά ομαδοποίηση.

Table of Contents

Abstract

Εκτεταμένη Περίληψη

List of Figures **iv**

List of Tables **vi**

List of Algorithms **viii**

Glossary **ix**

1 Introduction **1**

1.1 Machine Learning . . . 1

1.2 Generative Models . . . 5

1.2.1 Types of Generative Models . . . 5

1.2.2 Implicit Generative Models . . . 6

1.3 Data Clustering . . . 7

1.3.1 Clustering Framework . . . 9

1.3.2 K-means Algorithm . . . 10

1.4 Thesis Contribution . . . 12

1.5 Thesis Outline . . . 12

2 Deep Clustering **13**

2.1 Introduction . . . 13

2.2 Deep Clustering Framework . . . 14

2.3 Deep Neural Network Architecture . . . 15

2.4 Deep Features . . . 16

2.5 Non-Clustering Loss . . . 16

2.6 Clustering Loss 17
2.6.1 Principal Clustering Loss 18
2.6.2 Auxiliary Clustering Loss 18
2.7 Combining The Losses 19
2.8 Cluster Updates 20
2.9 Performance Evaluation Metrics 21

3 ClusterGAN **22**
3.1 Introduction 22
3.2 GANs 23
3.2.1 GANs Algorithm 24
3.3 ClusterGan 25
3.3.1 Sampling prior 25
3.3.2 Architecture 26
3.3.3 Loss function 27
3.3.4 ClusterGan Algorithm 27

4 Implicit Maximization of Likelihood **29**
4.1 Introduction 29
4.2 Imle Definition 30
4.3 Imle Algorithm 30
4.4 Nearest Neighbor Search 32
4.5 Imle Analysis 33
4.6 Implementing IMLE with a DNN 36

5 Deep Clustering Based on IMLE **37**
5.1 Introduction 37
5.2 Cluster-Friendly Sampling Prior 39
5.2.1 Mixture of Gaussians Distribution 39
5.2.2 Mixture of Discrete and Continuous Latent Variables 42
5.3 Two-Stage Nearest Neighbor Search 43
5.3.1 Training the Generator 44
5.3.2 Similarities with k-means 45
5.4 The Encoder's Network 46
5.5 The Objective Function 47

5.6 The proposed method 49

6 Experiments **51**

6.1 Datasets 51

6.2 Hyperparameter and Architecture Details 53

6.3 Experiments 57

6.4 Image Generation 66

6.5 Dimensionality Reduction 69

6.6 Training Procedure 71

7 Epilogue **73**

7.1 Conclusion 73

7.2 Suggestions for Future Work 73

Bibliography **75**

A Software requirements **82**

List of Figures

1.1 F. Rosenblatt's Perceptron. 3
1.2 Modern Deep Neural Network architecture with multiple layers. 4
1.3 Data clustering into two, four and six clusters. 8
1.4 K-means fails to cluster rings. 9

3.1 Basic GAN architecture and operation. 23
3.2 ClusterGan Architecture [1]. 26

4.1 4.1a The data points are represented by squares and the samples by circles. 4.1b For each data point the nearest sample is found. 4.1c Minimize the IMLE [3] objective 4.3. 32
4.2 Probability density of the model P_θ. 34
4.3 Cumulative distribution function as a function of distance. 34
4.4 Maximizing the cumulative distribution function (1st way). 35
4.5 Maximizing the cumulative distribution function (2nd way). 35
4.6 Generation of fake images by the use of IMLE [3]. 36

5.1 Input-output correspondence of a generative neural network. The 2-dimensional input is represented by z_0, z_1. The different colors represent the category of the generated sample produced by the neural network for the respective input (z_0, z_1). 38
5.2 Samples generated by a a mixture of Gaussians distribution. 40
5.3 5.3a Training results using a mixture of Gaussians distribution resulting in successful clustering. 5.3b Training results using a mixture of Gaussians distribution resulting in unsuccessful clustering. 42
5.4 IMLE Generator-Encoder Architecture. 47

6.1 Digits generated from distinct modes (MNIST with four classes). 67

6.2 Digits generated from distinct modes (Fashion-MNIST with five classes). 68
6.3 Latent Space Interpolation: MNIST (left) and Fashion-MNIST (right). . 68
6.4 Lowering Encoder's output manifold, using TSNE [4] at Fashion-MNIST dataset with five classes. 69
6.5 Lowering Encoder's output manifold, using TSNE [4] at MNIST dataset with four classes. 70
6.6 Lowering Encoder's output manifold, using TSNE [4] (perplexity=10) at Synthetic dataset with four classes. 70
6.7 Lowering Encoder's output manifold, using TSNE [4] (perplexity=30) at Synthetic dataset with four classes. 71
6.8 (a) Accuracy and normalized mutual information metrics based on centroid and encoder (b) Clustering error per epoch. 72

List of Tables

5.1 Similarities of the k-means algorithm and the modified IMLE algorithm. 46

6.1 The Generator's, Encoder's and Discriminator's architecture for the Synthetic dataset. 53
6.2 The Generator's, Encoder's and Discriminator's architecture for the Pendigits dataset with only odd numbers. 54
6.3 The Generator's, Encoder's and Discriminator's architecture for the Pendigits dataset with only even numbers. 54
6.4 The Generator's, Encoder's and Discriminator's architecture for the Pendigits dataset. 54
6.5 The Generator's, Encoder's and Discriminator's architecture for the MNIST and Fashion-MNIST datasets. 55
6.6 The Generator's, Encoder's and Discriminator's architecture for the MNIST datasets with four classes. 56
6.7 The Generator's, Encoder's and Discriminator's architecture for the fashion-MNIST datasets with five classes. 56
6.8 Presentation of different algorithmic variations used. 57
6.9 Presentation of the hyperparameters selection for each algorithm. . . . 58
6.10 Comparison of clustering metrics across datasets. 59
6.11 Comparison of clustering metrics across datasets. 60
6.12 Statistical analysis of the Accuracy on Synthetic Data. 61
6.13 Statistical analysis of the NMI on Synthetic Data. 61
6.14 Statistical analysis of the Accuracy on MNIST dataset. 61
6.15 Statistical analysis of the NMI on MNIST dataset. 62
6.16 Statistical analysis of the Accuracy on MNIST dataset with 4 classes. . . 62
6.17 Statistical analysis of the NMI on MNIST dataset with 4 classes. 62

6.18 Statistical analysis of the Accuracy on Fashion-MNIST. 63
6.19 Statistical analysis of the NMI on Fashion-MNIST dataset. 63
6.20 Statistical analysis of the Accuracy on Fashion-MNIST with 5 classes. . 63
6.21 Statistical analysis of the NMI on Fashion-MNIST with 5 classes. 64
6.22 Statistical analysis of the Accuracy on Pendigits. 64
6.23 Statistical analysis of the NMI on Pendigits. 64
6.24 Statistical analysis of the Accuracy on Pendigits with odd numbers. . . 65
6.25 Statistical analysis of the NMI on Pendigits with odd numbers. 65
6.26 Statistical analysis of the Accuracy on Pendigits with even numbers. . . 65
6.27 Statistical analysis of the NMI on Pendigits with even numbers. 66

List of Algorithms

1.1 Perceptron Learning Algorithm 5
1.2 k-means 11
3.1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps k update the discriminator is a hyperparameter.[2] 24
3.2 Minibatch stochastic gradient descent training of ClusterGan [1]. 28
4.1 Implicit maximum likelihood estimation (Imle) procedure [3] 31
5.1 Generating samples through a mixture of Gaussians distribution. 41
5.2 The two-step nearest neighbor search algorithm. 44
5.3 Clustering via the IMLE model and two-stage nearest neighbor search. 45
5.4 Deep Clustering Based on IMLE 50

Glossary

ACC	(Clustering) Accuracy
AI	Artificial Intelligence
ANN	Artificial Neural Network
CNNs	Convolutional Neural Networks
D	Discriminator (Network)
DBNs	Deep Belief Networks
DL	Deep Learning
DNNs	Deep Neural Networks
G	Generator (Network)
GANs	Generative Adversarial Networks
IMLE	Implicit Maximum Likelihood Estimation
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLP	Multilayer Perceptron
NMI	Normalized Mutual Information
SGD	Stochastic Gradient Descent
VAE	Variational Autoencoder
i.i.d.	independent and identically distributed

Chapter 1

Introduction

1.1 Machine Learning

1.2 Generative Models

1.3 Data Clustering

1.4 Thesis Contribution

1.5 Thesis Outline

1.1 Machine Learning

As soon as electronic computers came into use in the 1950s and 1960s, researchers have wondered if it was possible to program them, to learn and improve automatically by experience. If that was possible, the impact would be dramatic, not only in computer science but also in every field of science in general. This question gave birth to Machine Learning (ML). Learning is one of the most fundamental requirements for any type of intelligent behavior, and this is a common belief among the researchers. We can define ML as the field of computer science that aims, as its name implies, to create intelligent machines that automatically improve with experience by the use of data. Therefore, ML is one of the major branches of artificial intelligence (AI), and indeed, it is one of the most rapidly developing subfields of AI research [5].

From the very beginning, three main branches of machine learning emerged. Classical work in symbolic learning is described by E. B. Hunt [6], in statistical methods by N. J. Nilsson [7], as well in neural networks by F. Rosenblatt [8]. Through the years, all three branches developed advanced methods [9]: statistical or pattern recognition methods, such as the k-nearest neighbors, discriminant analysis, Bayesian classifiers, inductive learning of symbolic rules, such as top-down induction of decision trees, decision rules, and induction of logic programs, artificial neural networks, such as the multilayered feed-forward neural network with backpropagation learning, the Kohonen's self-organizing network, and the Hopfield's associative memory.

The origin of machine learning in its modern sense is usually more associated with the name of the psychologist F. Rosenblatt from Cornell University, who, based on ideas about the work of the human nervous system, created a group that built a machine for recognizing the letters of the alphabet Rosenblatt [10]. The machine, called the "Perceptron" by its creator, used both analog and discrete signals, and it also included a threshold element that converted analog signals into discrete ones. It is considered the first artificial neural network (ANN), and it was invented in 1958 by F. Rosenblatt. This work became the prototype of modern artificial (deep) neural networks (DNNs) that are used today.

Over the years, at the same time with machine learning research, vast advancements in computer hardware took place. As a result the Central Processing Units (CPUs) and Graphics Processing Units (GPUs) became computationally powerful. The rapid technology improvement and the creation of the Internet gave birth to the enormous data databases required for ML methodologies. Under these conditions, the DNNs methodologies became more commonly used. Today DNNs methodologies became powerful and can solve complex problems like computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, material inspection, and board game programs, where they have produced results not only comparable to expert human performance but in some cases surpassing it too [11, 12, 13].

In figure 1.1 we present the Perceptron. The Perceptron algorithm consists of only a single neuron. At the first layer of the Perceptron neural network (input layer), the algorithm expects the d dimensional input data. Then each entry x_i is multiplied with its corresponding weight w_i. The w_0 weight is called bias, and it is always multiplied by the number one. In the summation unit, the perceptron computes the following equation 1.1.

$$u(x) = w_0 + \sum_{i=1}^{d} x_i w_i \tag{1.1}$$

Afterward, the total sum passes through the activation function $\phi(u)$, defined as follows:

$$\phi(u) = \begin{cases} 1, & \text{if u(x)} > 0 \\ \text{-1}, & \text{otherwise} \end{cases} \tag{1.2}$$

Finally, at the output level the perceptron output equals to $o = \phi(u)$.

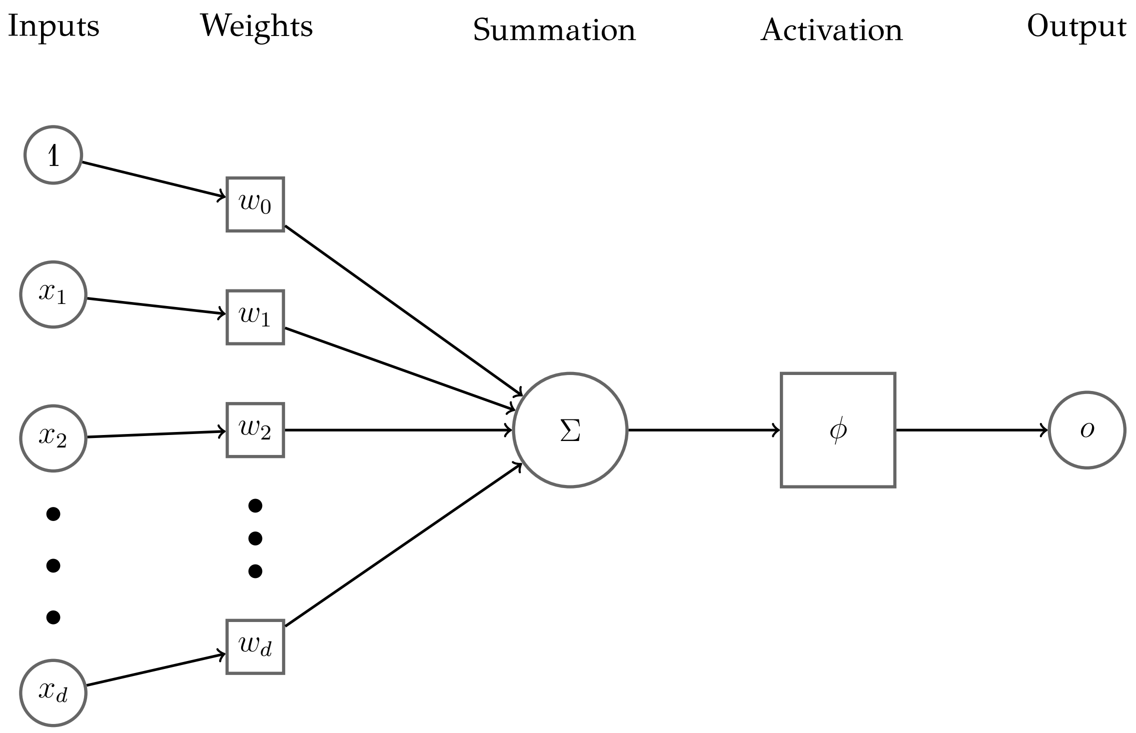


Figure 1.1: F. Rosenblatt's Perceptron.

In figure 1.2 we present a type of modern feedforward Deep Neural Network architecture which is called Multilayer Perceptron (MLP), which contains three hidden layers. The MLP consists of multiple layers which are fully connected, meaning that each node in one layer connects with a specific weight w_{ij} to every node in the following layer. It is called MLP because each node in the hidden layers and the output layer is a Perceptron with a non-linear activation function. In the input layer, the MLP expects the d dimensional input data. Thus, the input data pass through each network layer from the input layer to the output layer.

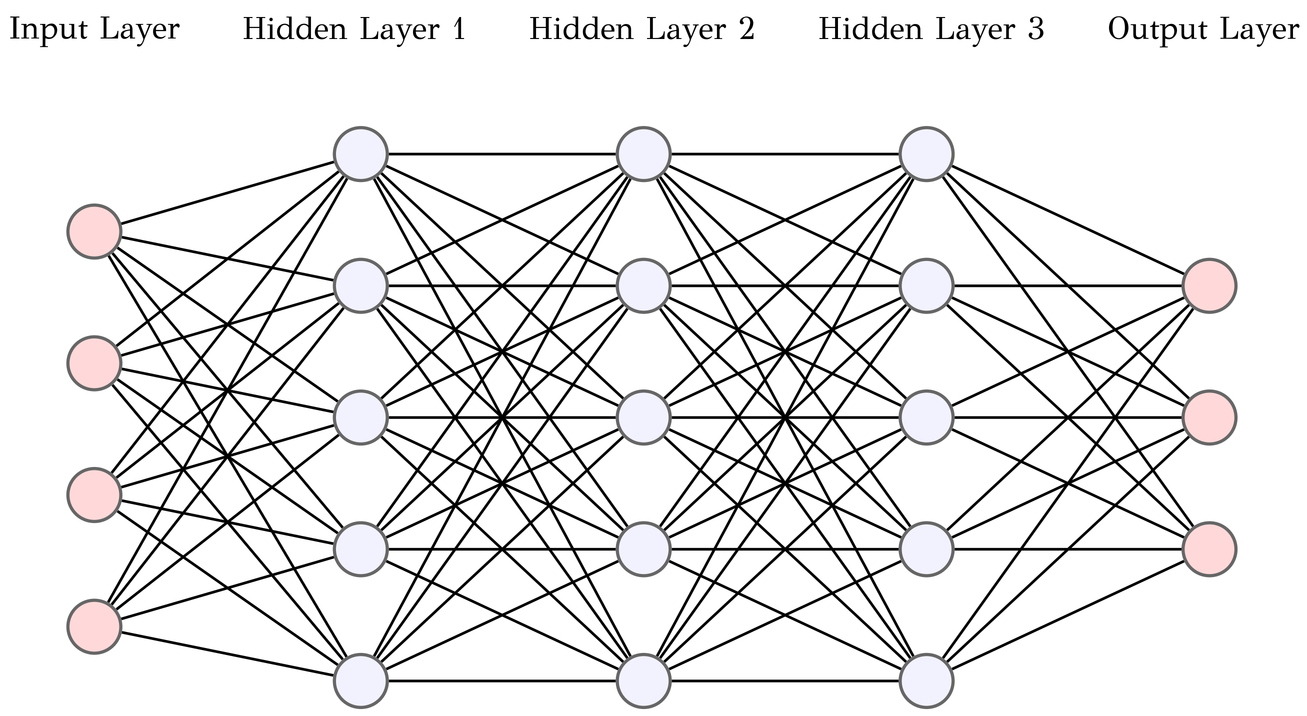


Figure 1.2: Modern Deep Neural Network architecture with multiple layers.

The valuable information is hidden in the network weights. In the case of Perceptron, in order to learn the optimal weights (or parameters) for the 2-class classification problem, it is necessary to train the Perceptron using data examples as an input to the algorithm. The Perceptron is a linear classifier and converges when all the inputs are classified correctly. This means that if the training set is linearly separable, then the Perceptron is guaranteed to converge [14] but if not then, the Perceptron will not converge, and the learning process will fail. In algorithm 1.1 we present the Perceptron's learning procedure.

In the case of feed-forward DNNs, in order to learn the optimal weights (or parameters) for the multiclass classification problem, more advanced learning procedures are required. Backpropagation is a widely used algorithm for training these kinds of deep architectures. In fitting a DNN, backpropagation computes the gradient of the loss function with respect to the weights of the whole network for a single input-output example. Furthermore, backpropagation is efficient and avoids the direct computation of the gradient with respect to each weight individually. This efficiency makes it feasible to use gradient-based methods for training DNNs and updating their weights in order to minimize the loss function. Some of the most popular gradient-based algorithms are gradient descent, and its variants such as stochastic gradient descent

[15, 16]. The backpropagation algorithm works by computing the gradient of a loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule.
The Perceptron Learning Algorithm follows 1.1:

Algorithm 1.1 Perceptron Learning Algorithm

```
Require: $P \leftarrow$ inputs with label 1.
Require: $N \leftarrow$ inputs with label 0.
Require: $\eta > 0$, which is defines the learning rate.
Ensure: Initialize $w$ randomly.
 1: while not converage do
 2:     Pick random $x \in P \cup N$
 3:     if $x \in P$ and $w^T x < 0$ then
 4:         $w \leftarrow w + \eta x$
 5:     end if
 6:     if $x \in N$ and $w^T x \geq 0$ then
 7:         $w \leftarrow w - \eta x$
 8:     end if
 9: end while
10: return $w$
```

1.2 Generative Models

1.2.1 Types of Generative Models

It is useful to distinguish between two types of probabilistic models, the prescribed models and the implicit models [17]. On the one hand, the direct probability models are those that provide a precise parametric specification $q_\theta(x)$ of the distribution of an observed random variable x and define a log-likelihood function $\log q_\theta(x)$ with parameters θ. Traditional models of machine learning and statistics are like this. On the other hand, implicit probabilistic models define a stochastic process that directly generates data, but the function $q_\theta(x)$ is not easy to define.

The implicit generative models use a latent variable $z \in \mathbb{R}^m$ that they sample from a prior distribution and they transform it with a deterministic function G_θ ($\mathbb{R}^m \to \mathbb{R}^d$) making use of the θ parameters. DNNs usually implement this kind of implicit generative model. In general, implicit generative models define the following marginal probability:

$$q_\theta(x) = \frac{\partial}{\partial x_1} \dots \frac{\partial}{\partial x_d} \int_{\{G_\theta(z) \le x\}} q(z)dz; \qquad x = G_\theta(z^{'}); \qquad z^{'} \sim q(z) \tag{1.3}$$

where $q(z)$ is a latent variable that provides the external source of randomness, and the relation 1.3 defines the transformed density as the derivative of the cumulative distribution function. When the function G_θ is well defined so that it is invertible, or for its dimensions, it holds that $m = d$ with easy characteristic roots, we use the rule for transformations of probability distributions.

It is interesting to develop more general and flexible implicit generative models where the G is a non-linear function with $d > m$ dimensions, defined by deep neural networks. In this case, the integral 1.3 is very difficult to calculate, and attempting to reduce this to a closed-form expression is hopeless. Evaluating it numerically is also challenging since the domain of integration could consist of an exponential number of disjoint regions, and numerical differentiation is ill-conditioned [3]. Even in direct generative models, the integral 1.3 is challenging to calculate. However, in the case of implicit generative models, the lack of the likelihood term reduces the available options for learning the model. In implicit generative models, this difficulty has created the need for methods that go beyond the lack of the likelihood difficulty 1.3, or they avoid the problem by being likelihood-free.

1.2.2 Implicit Generative Models

In this thesis, two types of implicit generative models are studied. The first is the implicit maximum likelihood estimation model (IMLE) [3] while the second one is the generative adversarial network (GANs) [2]. Both of them are used for data generation and can be used for data clustering. We name the first model, which used IMLE for deep clustering, Deep Clustering Based on IMLE, while the second one which uses GANs is the ClusterGan [1]. Both of these generative approaches are implicit models, often referred to as generative models. On the one hand, the method of IMLE [3] is based on the idea that in the area where there are training examples, the density of the model's probability distribution should be high. On the other hand, GANs [2] implement a learning process based on the distinction between real and generated (fake) data through two competing networks. In both cases, these models are trained without the formulation of a likelihood term.

1.3 Data Clustering

Organizing data into sensible groups is one of the most fundamental ways for understanding and learning. Clustering is one type of unsupervised learning and is defined as a process of assigning objects to groups, i.e., dividing data into subsets, clusters, as they are known in the literature [18, 19, 20], so that the data share common characteristics. Therefore, the main goal of clustering is for objects belonging to the same group to be similar (or related) to each other, as well as to differ (or not be related) to objects in other groups [21]. This way, clustering explores the data and aims to find (hidden) structures in them. At the same time, clustering is one of the most challenging problems in the field of ML. This is because clustering does not use category labels that tag objects with prior identifiers, i.e., class labels, and this is why it is categorized as an unsupervised learning problem. Clustering has been studied in many areas such as machine learning, data mining, pattern recognition, image analysis, data compression, computer graphics, information retrieval, statistics, and bioinformatics. It is also applicable in other fields of science, such as biology, psychology, and medicine [22, 23].

In a more mathematical formulation, clustering can be described as follows: Suppose a set of n data points $X = \{x_1, x_2, ..., x_n\}$ be the dataset, where $x_i \in \mathbb{R}^d$, and a k-clustering of X is defined as the partition of X into k clusters $\{C_1, C_2, ..., C_k\}$, so that the following three conditions are satisfied:

- $C_i \neq \varnothing, \quad i = 1, ..., k$
- $\bigcup\limits_{i=1}^{k} C_i = X$
- $C_i \cap C_j = \varnothing, \quad i \neq j, \quad i, j = 1, ..., k$

In addition, the data points contained in a cluster C_i are more similar to each other and less similar to the data points of the other clusters [24].

In many applications, the concept of a cluster is abstract [21]. To understand the difficulty better in defining the composition of a cluster, let us emphasize at the figure

1.3. The example shows 20 points and three different ways of clustering them. In the figure 1.3(b) and 1.3(d) we see the separation of the data into two and six groups respectively. It can also be argued that the points form four groups as shown in figure 1.3(c). Therefore, it is evident that the definition of a group is vague, and grouping depends on the nature of the data and the purpose of the analysis.

Evaluating the performance of clustering is also a tough question. This is because clustering aims at uncovering hidden structures of the data and defining an optimization criterion seems a non-trivial task. If the underlying data structure does not obey the optimization criterion, then the clustering algorithm will probably fail [25]. For instance, in figure 1.4, we can clearly see that although the correct number of clusters was selected, the k-means algorithm completely failed to capture the internal structure of the data, since the solution returned has no quality value.

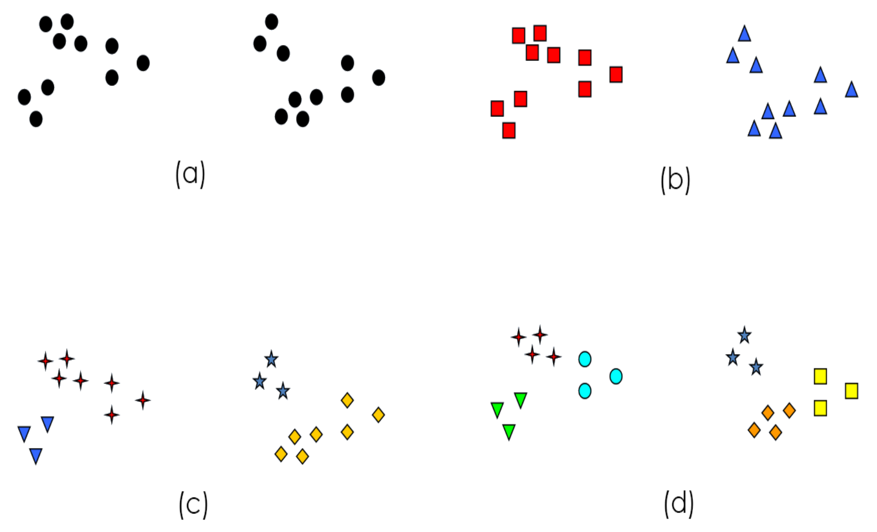


Figure 1.3: Data clustering into two, four and six clusters.

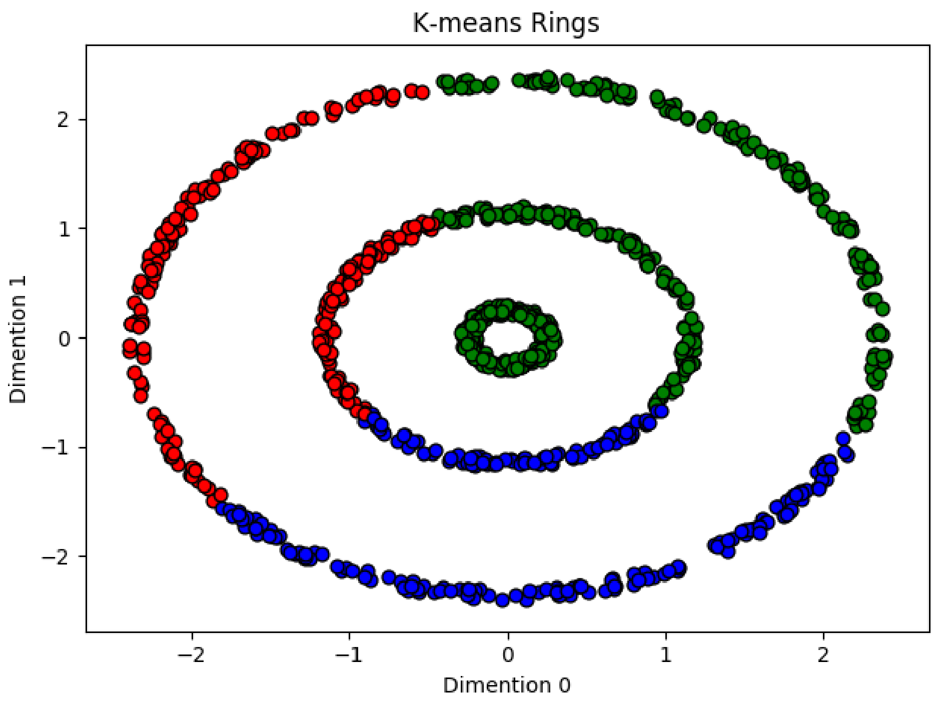


Figure 1.4: K-means fails to cluster rings.

1.3.1 Clustering Framework

Typically the clustering process includes the following [20]:

- **Pattern representation**: it refers to the number of classes, the number of available patterns, and the number, type, and scale of the features available to the clustering algorithm. Optionally includes feature extraction and/or selection.
- **Proximity measure**: definition of a pattern proximity measure that is appropriate to the data domain. The similarity of the data points is calculated with some distance metric. The most common distance metric is Euclidean Distance.
- **Clustering procedure**: the most common clustering output is hard or fuzzy partition of the dataset. Hard clustering category divides the data into groups, and each data point belongs to one cluster. On the other hand, a fuzzy clustering divides every data point into all data clusters with a degree of membership.
- **Evaluation**: all clustering algorithms will generate some data clustering, but this does not mean that the solution given is representative of the dataset. For this

reason, the data themselves must be first evaluated and then the algorithm and its results.

1.3.2 K-means Algorithm

Clustering has a long and rich history in a variety of scientific fields. One of the most popular and straightforward clustering algorithms, k-means, was first published in 1955. The k-means algorithm is one of the simplest and yet one of the most effective clustering procedures [21]. It belongs to the category of centroid-based clustering methods, which means that every cluster is characterized and represented by a central vector (centroid). Data points close to these vectors are assigned to the respective clusters. As an input k-means algorithm expects the dataset and the number of clusters (k clusters). The main goal of the algorithm is to define k centroids, one for each of the k clusters. Every data point is then assigned to its nearest centroid, and then each cluster centroid is updated based on the new points assigned to the cluster. This iterative 2-step procedure is repeated until none of the data points can change clusters or equivalently, all the centroids remain unchanged. Even though k-means was proposed over 50 years ago and many other algorithms have been proposed since then, K-means is still widely used. This reveals the difficulty in designing a general-purpose clustering algorithm, and the ill-posed problem of clustering [23].

Let's assume $X = \{x_1, ..., x_n\}$ be the dataset, where $x_i \in \mathbb{R}^d$, then clustering the dataset into K clusters $C_1, ..., C_K$ aims to separate the dataset in order to optimize a clustering criterion. To evaluate the clustering quality, the sum of the squared Euclidean distances is used between each data point x_i and the centroid m_k of the subset C_k which contains x_i. This criterion is called clustering error, and is one of the most common criteria for evaluating clustering. It depends on the centroids $m_1, ..., m_k$:

$$E(m_1, ..., m_k) = \sum_{i=1}^{N} \sum_{k=1}^{K} I(x \in C_k) ||x_i - m_k||^2 \tag{1.4}$$

where $I(x) = 1$ if x is true and 0 otherwise [26]. K-means performs the clustering procedure by optimizing the clustering error. If we had to choose between different data clusterings (supposing the same number of clusters) of a specific dataset, choosing the one with the lowest clustering error would be optimal. Minimizing this objective function is known to be an NP-hard problem even when k is equal to two ($k = 2$) [27]. Thus k-means, which is a greedy algorithm, typically converges to a local minimum.

It is worth mentioning that the k-means algorithm is used in a wide range of applications for its simplicity and also for its speed. However, a significant drawback of the method is the dependence on the initialization of the cluster centers. Therefore, in terms of performance, k-means does not guarantee that it will converge on the optimal solution in terms of clustering error. For this reason, solutions that approach the optimal are usually achieved after multiple runs of the algorithm with different initializations of the centers. Next we present the k-means algorithm:

Algorithm 1.2 k-means

Require: Dataset $D = \{(x^n)\}$, $x^n = (x_{n1}, ..., x_{nd})^T$, $n = 1, ..., N$, where N is the number of data, d is the dimentionality of data, and K is the selected number of clusters.

Ensure: Random initialization of the M centroids, with every centroid O_j to be a vector $\mu_j = (\mu_{j1}, ..., \mu_{jd})^T$ which contains the coordinates of the center of its cluster.

```
 1: $t \leftarrow 1$
 2: while true do
 3:     while $x_i^n$ unchecked in $x^n$ do
 4:         Compute the Euclidean distance $d(x_i^n, \mu_m)$ of $x_i^n$ from all centroids $\mu_m$.
 5:         Assert $x_i^n$ data point to $O_j$ cluster which its $\mu_j$ centroid is the nearest, meaning
            that: $d(x_i^n, \mu_j) = \min_K d(x_i^n, \mu_K)$.
 6:     end while
 7:     For every cluster $j$ compute the new centroid $\mu_j(t+1)$ as the mean of its data
        points.
 8:     if any of the centroid vectors $\mu_j$ changed then
 9:         $t \leftarrow t + 1$
10:     else
11:         Return the centroids $\mu_j$.
12:         Finish.
13:     end if
14: end while
```

1.4 Thesis Contribution

Deep Clustering is a new promising area of clustering algorithms that emerged in recent years. The main goal of Deep Clustering is creating clustering methodologies merged with DNNs architectures to exploit their representational power. Therefore, in this thesis, we will focus onf the new machine learning area of Deep Clustering and why it is considered promising. Afterward, we will present two Deep Clustering algorithms that were studied. The first Deep Clustering algorithm that we will discuss is the ClusterGan [1], which makes use of a modified GANs' [2] architecture in order to cluster the data in latent space $\mathcal{Z}$. The second Deep Clustering method that we will present is our contribution, and it is based on a generative DNN model that is trained by Implicit Likelihood Maximization (IMLE) [3]. IMLE is a creative way of maximizing the likelihood of the data indirectly. The Deep Clustering methodology based on IMLE also clusters the data in the latent space. Finally, we will analyze the experiments that took place and present the experimental results.

1.5 Thesis Outline

The following is a brief description of the structure of the next chapters:

- Chapter 2: Presentation of the relatively new area of ML, Deep Clustering. Additionally, a general Deep Clustering framework is described in detail.
- Chapter 3: Detailed description of the first model of the study, the ClusterGan [1], which performs deep clustering based on an adversarial learning scheme.
- Chapter 4: Detailed presentation of generative model training based on Implicit Maximum Likelihood Estimation (IMLE) [3].
- Chapter 5: Detailed presentation of the proposed Deep Clustering method that is based on IMLE [3].
- Chapter 6: Description of the experimental procedures that took place and analysis of experimental results.
- Chapter 7: Overview of the thesis and proposals for future research and improvements on Deep Clustering.

Chapter 2

Deep Clustering

2.1 Introduction

2.2 Deep Clustering Framework

2.3 Deep Neural Network Architecture

2.4 Deep Features

2.5 Non-Clustering Loss

2.6 Clustering Loss

2.7 Combining The Losses

2.8 Cluster Updates

2.9 Performance Evaluation Metrics

2.1 Introduction

Deep Learning (DL) is a powerful tool, which can learn rich and useful data representations from big data collections without relying heavily on human-engineered features [12, 28]. Many deep neural networks (DNNs) architectures rely primarily on the unsupervised learning stage, referred to as unsupervised pretraining (e.g., autoencoders). As a result, DNNs can learn better deep representations/features of the data. It is known that this methodology drastically improves the results of supervised learning tasks.

Although DL started mainly for supervised tasks, lately, it has found success in several unsupervised learning fields, like clustering, dimensionality reduction, etc. One of the main reasons why clustering with deep learning is a beneficial and interesting methodology is the fact that it has exploited the representational power of DNNs for preprocessing clustering inputs to improve the quality of clustering results. Using DNNs in this way, we can transform the data into more cluster-friendly spaces. This can be achieved due to their inherent property of highly nonlinear transformations. Existing popular clustering methods, such as k-means or spectral clustering, use either raw or linear transformed data. However, this would be insufficient when we are dealing with most datasets that have complex statistical properties.

Clustering has not always been the primary goal of DL, but since DL can provide rich and robust deep representation, it is reasonable to apply it in the clustering field [29, 30]. For simplicity of description, we will refer to clustering methods with DL as Deep Clustering in this thesis. More specifically, we can define deep clustering as a family of clustering methods that adopt DNNs to learn cluster-friendly representations [30].

2.2 Deep Clustering Framework

The most successful methods in deep clustering follow the same principles: representation learning using DNNs and using these representations as input for the clustering procedure. A general framework for deep clustering requires to define the following [31]:

- Deep Neural Network Architecture
- Deep Features used for Clustering
- Non-Clustering Loss
- Clustering Loss
- Method of combining the losses
- Cluster Updates

2.3 Deep Neural Network Architecture

In most deep clustering methods, the main part of the DNN is used to transform the inputs into a latent representation used for clustering. The following DNN architectures have previously been used for this goal:

- **Multilayer Perceptron (MLP)**: type of feedforward network, consisting of at least one hidden layer of neurons with no linear activation function, such that the output of every layer is the input to next one [32].
- **Convolutional Neural Network (CNN)**: was inspired by the biological process, where the connectivity pattern between neurons follows the organization of the animal visual cortex. CNN is an MLP explicitly designed to recognize regular-grid data such as images with a high degree of invariance to translation, scaling, skewing, and other forms of distortion [33, 12].
- **Deep Belief Network (DBN)**: is a probabilistic generative graphical model that is composed of multiple layers of stochastic, latent variables, which learn to extract a deep hierarchical representation of the input data [34]. It is composed of several stacked Restricted Boltzmann Machines (RBM) [35], such that the hidden layer of each sub-network serves as the visible layer of the next sub-network [36, 31].
- **Generative Adversarial Network (GAN)**: the architecture of two competing neural network models, the generator (G) and the discriminator (D), that engage in a zero-sum game. The G learns a distribution of interest to produce samples. The D aims to distinguish between real and generated samples. Competition in this game drives both networks to improve at their tasks until the generated samples are indistinguishable from the real data [2].
- **Variational Autoencoder (VAE)**: a generative model that learns the data distribution performing dimensionality reduction. Its architecture consists of three main parts, the encoder network, the latent space, and the decoder network [37].

2.4 Deep Features

Making use of the DNN architecture, the (deep) features that are used for clustering can be taken from one or more layers of the DNN:

- **One layer**: in this case, only one layer of the DNN is used to extract the deep features. Usually is beneficial because of its low dimensionality.
- **Several layers**: refers to the case where the deep features are extracted from a combination of the outputs of several layers. This way, the representation is richer and allows the embedded space to represent more complex semantic representations, which are capable of yielding better results in the similarity computation [38].

2.5 Non-Clustering Loss

The non-clustering loss comes purely from learning a deep representation of the data using DL methods, and it is independent of the clustering part of the procedure. Some possible options for non-clustering losses are [31]:

- **Absence of non-clustering loss**: in this case, the network model is only constrained by the clustering loss. The absence of a non-clustering loss can result in worse representations or even collapsing clusters [39].
- **Reconstruction loss**: in the case of selecting an autoencoder as DNN architecture, then the non-clustering loss is the reconstruction loss. Usually the reconstruction loss is a distance measure $d_{AE}(x_i, \hat{x}_i)$ between the input x_i to the autoencoder and the corresponding reconstruction $\hat{x}_i = f(x_i)$. The most common formulation of this is the mean squared error of the two variables:

  $$\mathcal{L}(x_i, f(x_i)) = d_{AE}(x_i, f(x_i)) = \sum_{i}^{n} ||x_i - f(x_i)||^2 \tag{2.1}$$

  where n is the number of data, x_i is the input and $f(x_i)$ is the autoencoder reconstruction.
- **Implicit maximum likelihood loss**: is the loss used in the current thesis in order to maximize the likelihood of the proposed model. The main goal of this

loss is to maximize the likelihood of the model overcoming the challenges of the direct likelihood maximization when the underlying machine learning model used is a DNN [3]. The implicit maximum likelihood loss follows:

$$\mathcal{L}(x_i, \widetilde{x}_i^{\theta}) = \sum_{i}^{n} ||x_i - \widetilde{x}_i^{\theta}||_2^2 \tag{2.2}$$

where n is the number of data, x_i is the i-th data point and $\widetilde{x}_i^{\theta}$ is the nearest sample (from m samples generated by the DNN model) to the i-th data point, using a distance metric (most common is the Euclidean distance). An extensive analysis of implicit maximum likelihood estimation is described in chapter 4.

- **The min-max loss**: if the model of choice is a GAN architecture [2] then the non-clustering loss function is the following:

$$\mathcal{L}(\mathcal{D}, \mathcal{G}) = \min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{data}(x)}[\log \mathcal{D}(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - \mathcal{D}(\mathcal{G}(z)))] \tag{2.3}$$

- **Additional information** about data can be used as a non-clustering loss in order to extract meaningful features, even if the extra information is not suitable for the clustering procedure.

2.6 Clustering Loss

Clustering loss functions guide the networks to learn cluster-friendly representations of the input data. Therefore such functions are called clustering loss functions [29, 30, 31]. There are two kinds of clustering losses where we can call the first one principal clustering loss and the second one auxiliary clustering loss [30].

- **Principal Clustering Loss**: after DNN training with this type of clustering loss, the clusters can be obtained directly, which means that this type of clustering loss functions contain the cluster centroids and the data clustering. Some examples of those type of clustering losses includes k-means loss [24], cluster assignment hardening loss [40], agglomerative clustering loss [41], cluster classification loss [1, 42], nonparametric maximum margin clustering [43] etc.

- **Auxiliary Clustering Loss**: this category of clustering loss enforces the DNN to learn a more clustering-friendly representation, but on the contrary, it is not able to provide clustering solutions directly. This means that deep clustering methods that use auxiliary clustering loss require running a clustering method after the training of the DNN to obtain the clusters. Some of the auxiliary clustering losses used in deep clustering are locality-preserving loss [44], which enforces the DNN to preserve the local property of data embedding, group sparsity loss [44], which exploits block diagonal similarity matrix for representation learning, etc.

2.6.1 Principal Clustering Loss

The following are some options for **principal clustering loss functions** [30]:

- **No clustering loss**: Even if a DNN is trained only with non-clustering losses, the extracted deep features can be used for clustering after its training. For example, a DNN can transform the input data into a lower representation space, performing dimensionality reduction. Such a transformation could be beneficial for the clustering sometimes, but using a clustering loss usually yields better results [40, 39].
- **K-means loss**: k-means loss or clustering error, enforces the new representation to be cluster-friendly [39]. We already analyzed the clustering error and its equation 1.4. Minimizing clustering error with respect to the DNN parameters ensures that the distance between each data point and its assigned cluster center will be small. Then, applying k-means would result in better clustering quality.
- **Agglomerative clustering loss**: Agglomerative clustering merges two clusters with maximum affinity (or similarity) in each step until some stopping criterion is fulfilled [45].
- **Cluster classification loss**: Cluster assignments obtained during cluster updates can be used as "mock" class labels for a classification loss in an additional network branch in order to encourage meaningful feature extraction in all network layers [1, 42].

2.6.2 Auxiliary Clustering Loss

Some options for **auxiliary clustering loss functions** follows [30]:

- **Locality-preserving loss**: this loss target is to ensure the locality of the clusters by pushing nearby data points together [44]. The mathematical formulation is the following:

$$\mathcal{L}_{lp} = \sum_i \sum_{j \in N_k(i)} s(x_i, x_j) ||f(x_i) - f(x_j)||^2 \tag{2.4}$$

where $N_k(i)$ is the set of k nearest neighbors of the data point x_i, $s(x_i, x_j)$ is a similarity measure between the points x_i and x_j, and $f(\cdot)$ is the nonlinear transformation implemented by a DNN.

- **Group sparsity loss**: it is inspired by spectral clustering. In this methodology, the block diagonal similarity matrix is exploited for representation learning [46]. Group sparsity is itself an effective feature selection method. As an example, in [44] the hidden units were divided into G groups, where G is the assumed number of clusters. When given a x_i the obtained representation has the form $\{f^g(x_i)\}_{g=1}^G$. Thus the loss can be defined as follows:

$$\mathcal{L}_{gs} = \sum_{i=1}^{N} \sum_{g=1}^{G} \lambda_g ||f^g(x_i)|| \tag{2.5}$$

where $\{\lambda_g\}_{g=1}^G$ are the weights to sparisity groups, defined as

$$\lambda_g = \lambda \sqrt{n_g} \tag{2.6}$$

where n_g is the group size and λ is a constant.

2.7 Combining The Losses

Let's suppose that a clustering and a non-clustering loss fuction is used in a deep clustering procedure. It is important to have a method of combining these two losses effectively. The most common way of achieving that is the following formulation:

$$\mathcal{L}(\theta) = \alpha \mathcal{L}_c(\theta) + (1 - \alpha) \mathcal{L}_n(\theta) \tag{2.7}$$

where $\mathcal{L}_c(\theta)$ is the clustering loss, $\mathcal{L}_n(\theta)$ is the non-clustering loss, and $\alpha \in [0, 1]$ is a constant specifying the weighting between the two-loss functions. α is an additional hyperparameter for the DNN training. It is possible for α to change during the DNN training phase under some schedule. Scheduling methods to adjust α during training are expressed as [30]:

- **Joint training**: α is set to a constant value $(0 < \alpha < 1)$ and the DNN training is affected by both loss functions simultaneously.
- **Variable schedule**: α varies during the training phase based on a chosen schedule. For example, α can start with a low value and gradually can be increased in every training epoch.
- **Pre-training, fine-tuning**: at the first stage, α is set to be 0, and the DNN is trained using the non-clustering loss only. In the second stage, α is set to be one, and the DNN is retrained using only the clustering loss. This way, the DNN can be fine-tuned for clustering. Training the DNN with the clustering error only for long enough can lead to worse clustering results.

2.8 Cluster Updates

One common categorization of clustering methods is into hierarchical and partitional (centroid-based) approaches [20]. Hierarchical clustering aims to build a hierarchy of clusters and data points. On the other hand, partitional clustering groups the data and creates cluster centers. This way, metric relations are used to assign each data point into the cluster with the most similar center [31].

In the field of deep clustering, the two most dominant methods that have been used are Agglomerative clustering combined with DL [45], which is a hierarchical clustering method, and k-means combined with DL [40, 39, 47, 42], which falls into the category of partitional clustering.

If a centroid-based method is used during the DNN training, the clusters and the centroids are updated. The two most common methodologies for updating clusters and centroids are expressed as [31]:

- **Jointly updated with the network model**: Cluster assignments are formulated as probabilities. In this case, they can be included as parameters of the network and optimized via back-propagation.
- **Alternatingly updated with the network model**: Clustering assignments are updated in a separate step than the one where the network model is updated [40, 45].

2.9 Performance Evaluation Metrics

In order to evaluate the results of the deep clustering methods, two standard supervised evaluation metrics are extensively used in the literature. These are the metrics used in this thesis and assume that a ground truth clustering is available. For all algorithms, the number of clusters is set to the number of ground-truth categories [30].

The first supervised metric is Clustering Accuracy (ACC):

$$ACC = \max_{m} \frac{\sum_{i=1}^{n} I(y_i = m(c_i))}{n} \tag{2.8}$$

where $I(x) = 1$ if x is true and 0 otherwise, y_i is the ground-truth label, c_i is the cluster assignment generated by the deep clustering algorithm, and m is a mapping function which ranges over all possible one-to-one mappings between assignments and labels. This metric finds the best matching between cluster assignments from a clustering method and the ground truth. The optimal mapping function can be efficiently computed by the Hungarian algorithm [48].

The second one is Normalized Mutual Information (NMI) [49]:

$$NMI(Y, C) = \frac{I(Y, C)}{\frac{1}{2}[H(Y) + H(C)]} \tag{2.9}$$

where Y denotes the ground-truth labels, C denotes the clusters labels, I is the mutual information metric and H the entropy.

Chapter 3

ClusterGAN

3.1 Introduction
3.2 GANs
3.3 ClusterGan

3.1 Introduction

Generative Adversarial Networks (GANs) [2] are systems based on a min-max strategy where two networks are competing with each other in a zero-sum game. The first network, the generator (G), generates data and the second one, the discriminator (D), discriminates between fake and real data. GANs have obtained remarkable success in many unsupervised learning tasks, and unarguably, clustering is an important unsupervised learning problem. Additionally, the latent space of GANs provides dimensionality reduction and gives rise to novel applications. For example, perturbations in the latent space could be used to determine adversarial examples that further help build robust classifiers [50]. Compressed sensing using GANs [51] relies on finding a latent vector that minimizes the reconstruction error for the measurements. Generative compression is yet another application involving the GAN latent space [52]. In this chapter, we will first present the main framework of GANs. Afterwards, we will introduce the ClusterGan [1], a deep clustering methodology that is based on GANs architecture and makes use of the generator's latent space and also an additional network, the Encoder (E), to cluster the data via an unsupervised classification manner.

3.2 GANs

GANs [2] aim to match the real data distribution through implicit sampling and simultaneously provide a mapping from a latent space $\mathcal{Z}$ to the input space $\mathcal{X}$. In order to achieve that the generator creates samples $x = G(z, \theta_g)$, where usually z corresponds to $z \sim \mathcal{N}(0, \sigma^2 I)$. Then its adversary, the discriminator D, tries to distinguish between real data and synthetic ones constructed by the generator, computing the output $D(x, \theta_d)$, which is the probability of each sample to be real. More specifically, the generator's objective is to maximize the discriminator's error while the discriminator aims to minimize it. This iterative process ends when the discriminator can no longer distinguish the real from the fake samples. Hence, the objective function of the complete network is the following:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{data}(x)}[\log \mathcal{D}(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - \mathcal{D}(\mathcal{G}(z)))] \tag{3.1}$$

where p_{data} represents the distribution of real data, p_z represents the prior probability distribution of noise (usually a Gaussian distribution) which G uses to generate fake data, x and z represent the samples from each corresponding space, $\mathbb{E}_x$ and $\mathbb{E}_z$ represent the expected log-likelihood from the different outputs of both real and generated samples, $\mathcal{D}$ outputs a real number between 0 and 1 representing the probability for data being real ($\mathcal{D}(x) \to 1$) or fake ($\mathcal{D}(x) \to 0$) and finally the $\mathcal{G}$ function outputs the generated samples.

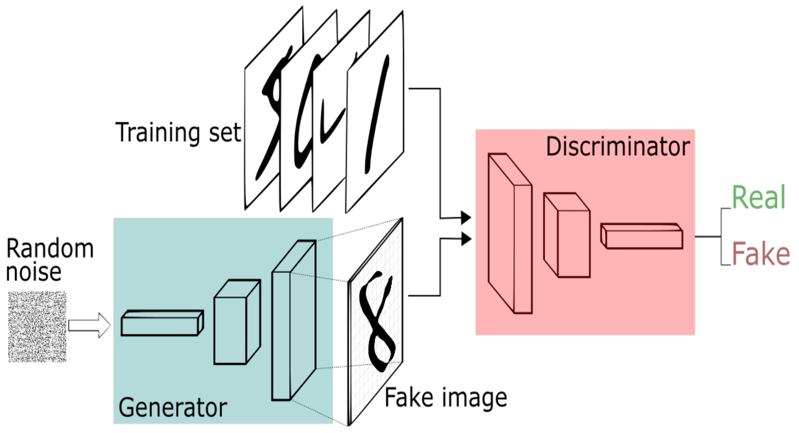


Figure 3.1: Basic GAN architecture and operation.

Figure 3.1 shows the architecture as well as the operation of a typical GAN. Initially, as an input to the generator's network, we introduce some random noise (usually sampled from a Gaussian distribution), and as an output, we obtain a fake sample. The discriminator's network takes as input real samples (from the actual dataset) and fake samples created by the generator. Finally, the discriminator's output is a real number $\in (0, 1)$ that assigns the category of the image as real ($\mathcal{D}(x) \to 1$) or fake ($\mathcal{D}(x) \to 0$).

3.2.1 GANs Algorithm

The complete training algorithm of GANs is presented below:

Algorithm 3.1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps k update the discriminator is a hyperparameter.[2]

```
1: $iteration \leftarrow 1$
2: while $iteration \leq ITERATIONS$ do
3:     $t \leftarrow 1$
4:     while $t < T$ do
5:         Sample minibatch of $m$ noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $p_g(z)$.
6:         Sample minibatch of $m$ examples $\{x^{(1)}, ..., x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
7:         Update the discriminator by ascending its stochastic gradient:
               $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log \mathcal{D}(x^{(i)}) + \log(1 - \mathcal{D}(\mathcal{G}(z^{(i)}))) \right].  \quad (3.2)$$
8:         $t \leftarrow t + 1$
9:     end while
10:    Sample minibatch of $m$ noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $p_g(z)$.
11:    Update the generator by descending its stochastic gradient:
               $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - \mathcal{D}(\mathcal{G}(z^{(i)}))).  \quad (3.3)$$
12:    $iteration \leftarrow iteration + 1$
13: end while
```

The gradient-based updates can use any standard gradient-based learning rule. The most popular choice is the Adam optimizer [16].

3.3 ClusterGan

The ClusterGan [1] is a clever way of using the GANs [2] methodology in order for achieving data clustering. In this section, we will present the main algorithmic ideas used in ClusterGan [1]. They are categorized as:

- **Sampling prior**: a mixture of discrete and continuous latent variables is used, which is able to create clusters more naturally in the latent space.
- **Architecture**: besides the Generator and the Discriminator network, the Encoder network is added, which is responsible for the clustering procedure. The Encoder provides an inverse-mapping from the space $\mathcal{X}$ to the embedded latent space $\mathcal{Z}$, $\mathcal{E} : \mathcal{X} \to \mathcal{Z}$.
- **Loss function**: jointly train the GAN along with the Encoder's inverse-mapping network with a clustering-specific loss. Therefore, the distance geometry in the projected space reflects the distance-geometry of the latent variables z.

3.3.1 Sampling prior

The first step of modifying the vanilla GAN [2] to improve the quality of clustering in the latent space is a better sampling prior distribution as input to the generator, which can create clusters more naturally in the latent space. More specifically, as an input to the generator's network, ClusterGan [1] utilizes a mixture of discrete and continuous latent variables to create a non-smooth geometry in the latent space. This is a useful property so that the constructed clusters in the latent space will be strictly separated from each other. This could be possible if we sample from a prior that consists of normal random variables cascaded with one-hot encoded vectors. To be more precise the latent variable is defined as $z = (z_n, z_c)$ where $z_n \sim \mathcal{N}(0, \sigma^2 I)$ and the latent vector z_c is sampled by an one-hot distribution of K elements (K is the number of clusters). In addition, the standard deviation σ is chosen in such a way that the one-hot vector provides sufficient signal to the ClusterGan [1] training that leads to each mode only generating samples from a corresponding class in the original data. More specifically, σ should be set to a small value like $\sigma = 0.10$ so that for each dimension of the normal latent variables, $z_{n,j} \in (-0.6, 0.6) << 1.0 \ \forall j$ holds with high probability. The choice of small value for standard deviation σ is crucial so that the clusters constructed in the latent space $\mathcal{Z}$ of the generator network will not overlap but remain separated.

3.3.2 Architecture

The ClusterGan [1] contains the typical GAN architecture and an additional network named Encoder. More specifically, the first network is the Generator which is responsible for generating fake samples based on the training data ($\mathcal{G} : z \rightarrow X_g$), the second network is Discriminator which classifies each sample as real, drawn from the dataset ($\mathcal{D} : X \rightarrow 1$), or fake, created by the Generator ($\mathcal{D} : X \rightarrow 1$). The Encoder is a third network that does not exist in vanilla GAN [2] architecture setups like the Generator and the Discriminator. The Encoder's addition is crucial for the clustering procedure, as its main function is to explicit inverse-map each generated (fake) sample from its original space back to the latent variable space ($\mathcal{E} : X_g \rightarrow \hat{z}$), in order to obtain the latent variables $\hat{z}$ given the data samples. This seems no easy task since the inverse-mapping problem creates a non-convex search space because the Generator's (mathematical) model is being implemented as a neural network and can provide different embeddings in the $\mathcal{Z}$ space based on initialization. Using the encoder network to explicit inverse-map the generated samples solves the problem efficiently. Through this procedure, the encoder network can cluster the samples in an unsupervised classification manner. For the purpose of understanding ClusterGan's methodology better, in figure (3.2) we mainly present ClusterGan's architecture and its basic operation. As an input the generator takes $z = (z_n, z_c)$ and creates fake samples x_g. The z_c part of the latent variable refers to the clustering procedure. The Discriminator as input receives the real (X_r) and the fake (X_g) samples and outputs their probability of being real data. Simultaneously the Encoder receives the generated sample and projects it back to the latent space.

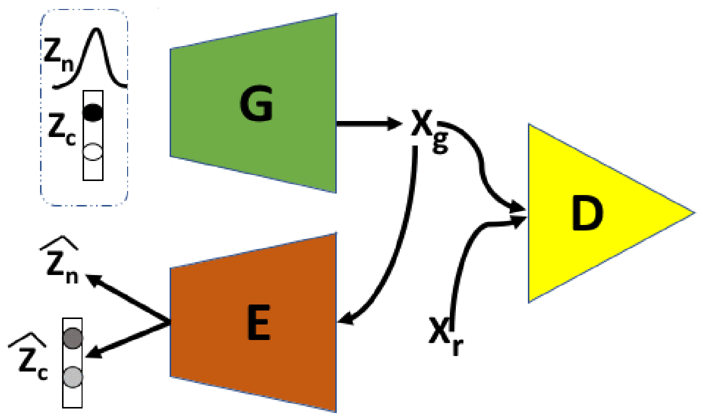


Figure 3.2: ClusterGan Architecture [1].

3.3.3 Loss function

The ClusterGan's loss function is a mixture of losses that contains the GANs' loss that targets the data generation and the clustering loss that aims at the data clustering. The objective function of the complete ClusterGan network is the following:

$$\begin{gathered}\min_{\theta_{\mathcal{G}},\theta_{\mathcal{E}}}\max_{\theta_{\mathcal{D}}} \mathop{\mathbb{E}}_{x\sim\mathbb{P}_x^r} q(\mathcal{D}(x)) + \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} q(1-\mathcal{D}(\mathcal{G}(z)))+ \\ \beta_n \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} ||z_n - \mathcal{E}(\mathcal{G}(z_n))||_2^2 + \beta_c \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} \mathcal{H}(z_c, \mathcal{E}(\mathcal{G}(z_c)))\end{gathered} \tag{3.4}$$

where $\mathcal{H}(.,.)$ is the cross-entropy loss, which is defined as $\mathcal{H}(X) = -\sum_x p(x)\log(p(x))$. The relative magnitudes of the regularization coefficients β_n and β_c enable a flexible choice to vary the importance of preserving the discrete and continuous portions of the latent code, where $q(.)$ is the quality function, given as $q(x) = log(x)$ for vanilla GAN [2], and $q(x) = x$ for Wasserstein GAN (WGAN) [53]. More specifically, the first two terms of the loss function equation 3.4 which are $\mathop{\mathbb{E}}_{x\sim\mathbb{P}_x^r} q(\mathcal{D}(x)) + \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} q(1 - \mathcal{D}(\mathcal{G}(z)))$ constitute the vanilla GANs' loss. The GANs' loss is necessary to ClusterGan's methodology for the data generation procedure. Without this part of ClusterGan's loss function, the data generation would not be possible setting the whole clustering procedure to fail. This is due to the fact that in order for the ClusterGan's method to produce a good quality data clustering in z latent space, it first must be able to generate samples. The last two terms of the loss function (equation 3.4), $\beta_n \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} ||z_n - \mathcal{E}(\mathcal{G}(z_n))||_2^2 + \beta_c \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} \mathcal{H}(z_c, \mathcal{E}(\mathcal{G}(z_c)))$, contain the clustering loss. The first part of the clustering loss, $\beta_n \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} ||z_n - \mathcal{E}(\mathcal{G}(z_n))||_2^2$, acts as a regularization of the continuous portion z_n of the latent space. This ensures the reversibility of the continuous portion z_n of the latent space given a sample $\mathcal{E} : \mathcal{G}(z_n) \to z_n$. The second part of the clustering error loss, $\beta_c \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} \mathcal{H}(z_c, \mathcal{E}(\mathcal{G}(z_c)))$, the discrete portion of the latent code. Its main goal is for all the mappings of points that belong in the same class in $\mathcal{X}$ space to have the same one-hot encoding when embedded in $\mathcal{Z}$ space. This regularization is the most important part to succeed in data clustering in the latent space. We can think of other variations of the regularization that map $\mathcal{E}(\mathcal{G}(z))$ to be close to the centroid of the respective cluster. For instance $||\mathcal{E}(\mathcal{G}(z)) - \mu^{c(i)}||_2^2$, in a similar way as K-Means.

3.3.4 ClusterGan Algorithm

The complete training algorithm of ClusterGan is presented below:

Algorithm 3.2 Minibatch stochastic gradient descent training of ClusterGan [1].

```
1: $iteration \leftarrow 1$
2: while $iteration < ITERATIONS$ do
3:     $t \leftarrow 1$
4:     while $t < T$ do
5:         Sample minibatch of $m$ noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $\mathbb{P}_z$, where $z = (z_n, z_c)$.
6:         Sample minibatch of $m$ examples $\{x^{(1)}, ..., x^{(m)}\}$ from data generating distribution $\mathbb{P}_x$.
7:         Update the discriminator by ascending its stochastic gradient:
               $$\nabla_{\theta_{\mathcal{D}}} \frac{1}{m} \sum_{i=1}^{m} \left[ q(\mathcal{D}(x^{(i)})) + q(1 - \mathcal{D}(\mathcal{G}(z^{(i)}))) \right]. \quad (3.5)$$
8:         $t \leftarrow t + 1$
9:     end while
10:    Sample minibatch of $m$ noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $\mathbb{P}_z$, where $z = (z_n, z_c)$.
11:    Update the generator by descending its stochastic gradient:
           $$\nabla_{\theta_{\mathcal{G}}} \frac{1}{m} \left( - \sum_{i=1}^{m} q(\mathcal{D}(\mathcal{G}(z^{(i)}))) + \beta_n \sum_{i=1}^{m} ||z_n^{(i)} - \mathcal{E}(\mathcal{G}(z_n^{(i)}))||_2^2 + \beta_c \sum_{i=1}^{m} \mathcal{H}(z_c^{(i)}, \mathcal{E}(\mathcal{G}(z_c^{(i)}))) \right). \quad (3.6)$$
12:    Update the encoder by descending its stochastic gradient:
           $$\nabla_{\theta_{\mathcal{E}}} \frac{1}{m} \left( \beta_n \sum_{i=1}^{m} ||z_n^{(i)} - \mathcal{E}(\mathcal{G}(z_n^{(i)}))||_2^2 + \beta_c \sum_{i=1}^{m} \mathcal{H}(z_c^{(i)}, \mathcal{E}(\mathcal{G}(z_c^{(i)}))) \right). \quad (3.7)$$
13:    $iteration \leftarrow iteration + 1$
14: end while
```

The gradient-based updates can use any standard gradient-based learning rule. The most popular choice is the Adam optimizer [16].

Chapter 4

Implicit Maximization of Likelihood

4.1 Introduction

4.2 Imle Definition

4.3 Imle Algorithm

4.4 Nearest Neighbor Search

4.5 Imle Analysis

4.6 Implementing IMLE with a DNN

4.1 Introduction

Consider a model distribution that maximizes the likelihood of the data. The likelihood is the product of densities evaluated at all data points, so the model density should be high at each data point. The mathematical formulation of the likelihood function is given by:

$$L(\theta) = \prod_{i=1}^{n} p(x_i; \theta) \tag{4.1}$$

where x_i is a random sample from a distribution that depends on unknown parameters θ with probability density function $p(x_i; \theta)$. Suppose that we cannot observe the model distribution directly as a closed-form function, but instead, we can only observe independent and identically distributed (i.i.d.) samples drawn from the model. Because the density at the data points is high, most samples are expected to lie near the data points. Therefore, a natural objective is to minimize the distance from each data point to the nearest sample [3].

4.2 Imle Definition

Suppose a given set of n data points $x_1, . . . , x_n$ and some unknown parameterized probability distribution P_θ with density p_θ. In addition, it is also possible to draw independent and identically distributed (i.i.d.) samples from P_θ. Let $\widetilde{x}_1^\theta, . . . , \widetilde{x}_m^\theta$ be i.i.d. samples from P_θ, where $m \geq n$. For each data point x_i, we define a random variable R_i^θ to be the distance between x_i and the nearest sample. More precisely,

$$R_i^\theta := \min_{j \in [m]} ||\widetilde{x}_j^\theta - x_i||_2^2 \tag{4.2}$$

where $[m]$ denotes $\{1, ..., m\}$.
The implicit maximum likelihood estimator $\hat{\theta}_{IMLE}$ [3] is defined as:

$$\hat{\theta}_{IMLE} := \arg\min_\theta \mathbb{E}_{R_1^\theta,...,R_n^\theta}\left[\sum_{i=1}^n R_i^\theta\right] = \arg\min_\theta \mathbb{E}_{\widetilde{x}_1^\theta,...,\widetilde{x}_m^\theta}\left[\sum_{i=1}^n \min_{j \in [m]} ||\widetilde{x}_j^\theta - x_i||_2^2\right] \tag{4.3}$$

4.3 Imle Algorithm

The Implicit Maximum Likelihood Estimator (IMLE) [3] procedure is outlined in Algorithm 4.1. For input the algorithm takes the dataset $D = \{x_i\}_{i=1}^n$ and a sampling mechanism for the implicit model P_θ with unknown parameters θ. First, we initialize the θ parameters of the model to random values. In each outer iteration, we draw m i.i.d. samples from the current model P_θ. Then we randomly select a batch of data points from the dataset and find the nearest sample from each data point. In each inner iteration, a standard iterative optimization algorithm is used, like stochastic gradient descent (SGD) [15], to minimize a sample-based version of the IMLE [3] objective.

Algorithm 4.1 Implicit maximum likelihood estimation (Imle) procedure [3]

```
Require: The dataset $D = \{x_i\}_{i=1}^{n}$ and a sampling mechanism for the implicit model $P_\theta$.
Ensure: Initialize $\theta$ to a random vector.
 1: $t \leftarrow 0$
 2: while $t < T$ do
 3:     Draw i.i.d. samples $\widetilde{x}_1^\theta, \ldots, \widetilde{x}_m^\theta$ from $P_\theta$.
 4:     Pick a random batch $S \subseteq \{1, ..., n\}$
 5:     $\sigma(i) \leftarrow \arg\min_j ||\widetilde{x}_j^\theta - x_i||_2^2, \forall i \in S$
 6:     $l \leftarrow 0$
 7:     while $l < L$ do
 8:         Pick a random mini-batch (mini-batch) $\widetilde{S} \subseteq S$
 9:         $\theta \leftarrow \theta - n\nabla_\theta(\frac{n}{|\widetilde{S}|} \sum_{i\in\widetilde{S}} ||\widetilde{x}_{\sigma(i)}^\theta - x_i||_2^2)$
10:         $l \leftarrow l + 1$
11:     end while
12:     $t \leftarrow t + 1$
13: end while
14: return $\theta$
```

In order to fully understand how the IMLE [3] algorithm works, we will analyze each major algorithmic step with a two-dimensional example. The basic idea can be summarized as follows. In every training epoch:

- Draw m i.i.d. samples from the current model P_θ.
- Find the nearest sample from each data point.
- Minimize the IMLE [3] objective function 4.3.

A representation of a two-dimensional example is shown in the following figures:

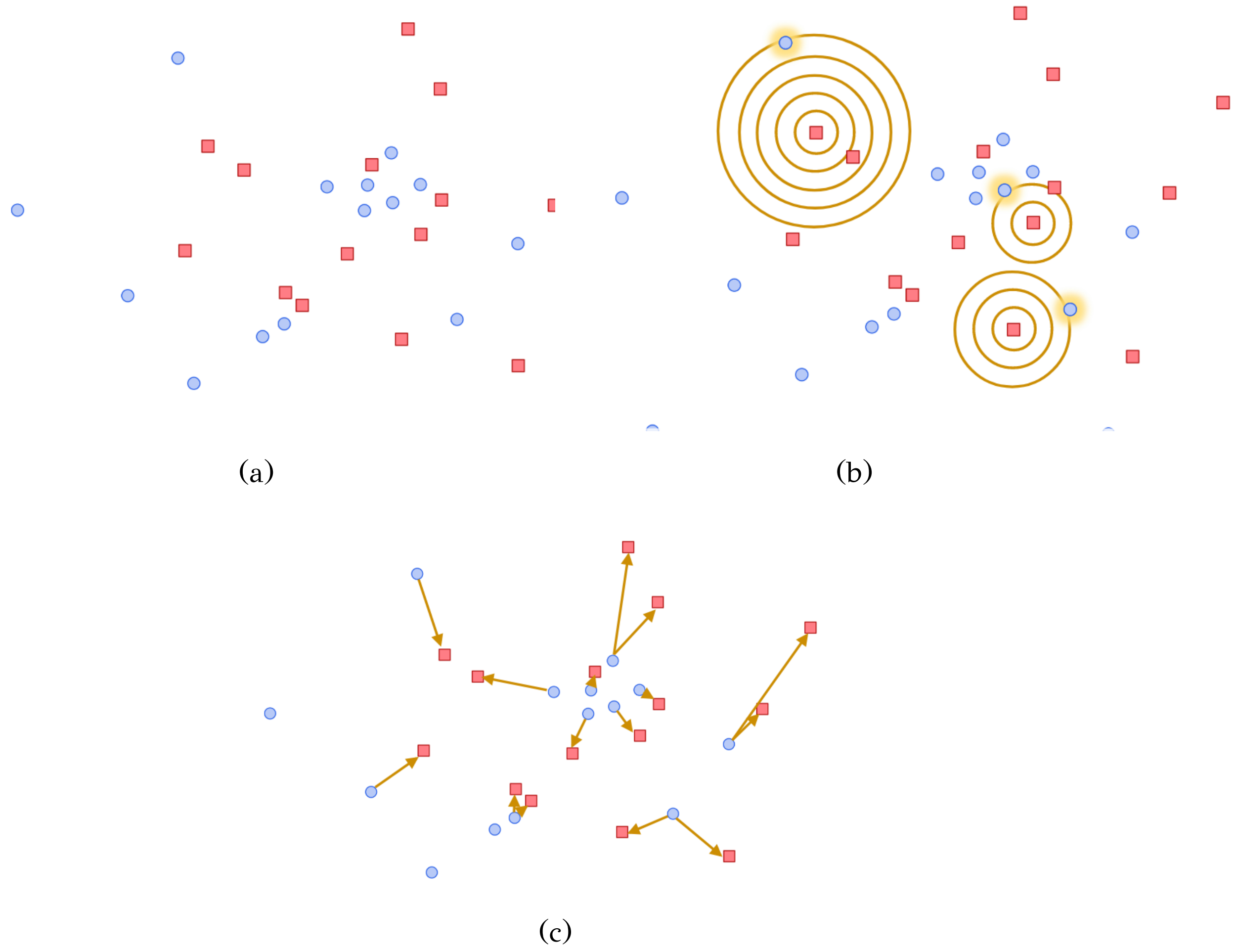


Figure 4.1: 4.1a The data points are represented by squares and the samples by circles. 4.1b For each data point the nearest sample is found. 4.1c Minimize the IMLE [3] objective 4.3.

4.4 Nearest Neighbor Search

In order to apply the IMLE algorithm [3], it is necessary to solve a nearest neighbor search problem at each iteration. This means that the scalability of the method depends on the ability to find the nearest neighbors quickly. This is considered to be a complex problem, especially in high dimensions. However, due to recent advances in nearest neighbor search algorithms [54, 55], which avoid the curse of dimensionality in time complexity that often arises in nearest neighbor search, this is no longer the case.

It is worth noting that the use of Euclidean distance is not a significant limitation of the algorithm. A variety of distance metrics are either exactly or approximately equivalent to Euclidean distance in some non-linear embedding space. The theoretical guarantees are inherited from the Euclidean case. This encompasses popular distance metrics used in the literature, like the Euclidean distance between the activations of a neural net, which is often referred to as a perceptual similarity metric [56, 57] and the method can be easily extended to use these metrics.

4.5 Imle Analysis

It turns out that the IMLE method is equivalent to the maximum likelihood estimator (MLE) under some conditions [3]. A simple hypothesis will be presented below to show why this is the case. For simplicity, let's consider the special case where there is a single data point x_1 and we generate a single sample $\widetilde{x}_1^\theta$. Consider the total density of P_θ inside a sphere of radius t centered at x_1 as a function of t. This function will be defined as $\widetilde{F}^\theta(t)$ (figure 4.2). On the one hand, if the density in the neighborhood of x_1 is high, then $\widetilde{x}_1^\theta$ would grow rapidly as t increases. On the other hand, if the density in the neighborhood of x_1 is low, then $\widetilde{F}^\theta(t)$ would grow slowly (figure 4.3). So, maximizing likelihood is equivalent to making $\widetilde{F}^\theta(t)$ grow as fast as possible. Therefore, we can maximize the area with respect to the function $\widetilde{F}^\theta(t)$ (figure 4.4), or equivalently, minimize the area considering $1 - \widetilde{F}^\theta(t)$ function (figure 4.5). $\widetilde{F}^\theta(t)$ function can be interpreted as the cumulative distribution function (CDF) of the Euclidean distance between x_1 and $\widetilde{x}_1^\theta$, where $\widetilde{x}_1^\theta$ is a random variable that will be denoted as $\widetilde{R}^\theta$. The function $\widetilde{R}^\theta$ is non-negative (as it counts distance), so its expected value comes from the following integral:

$$\mathbb{E}[\widetilde{R}^\theta] = \int_0^\infty Pr(\widetilde{R}^\theta > t)\mathrm{d}t = \int_0^\infty (1 - \widetilde{F}^\theta(t))\mathrm{d}t \tag{4.4}$$

which is exactly the area under the function $1 - \widetilde{F}^\theta(t)$ mentioned above. Therefore, in order to maximize the likelihood of the data point x_1, we can minimize $\mathbb{E}[\widetilde{R}^\theta]$, or minimizing the expected distance between the data point and a random sample.

In figure (4.2) we present the model's distribution. The vertical bar denotes the data point x_1. We also define a random variable $||\widetilde{x}_1^\theta - x_1||_2$, that indicates the Euclidean distance of sample $\widetilde{x}_1^\theta$ from the data point x_1. The cumulative distribution function $Pr(|| \widetilde{x}_1^\theta - x_1 ||_2 \leq t)$ increases as t increases. We expect by construction this function to grow rapidly. This is accomplished by moving the sample $\widetilde{x}_1^\theta$ closer to the data point x_1.

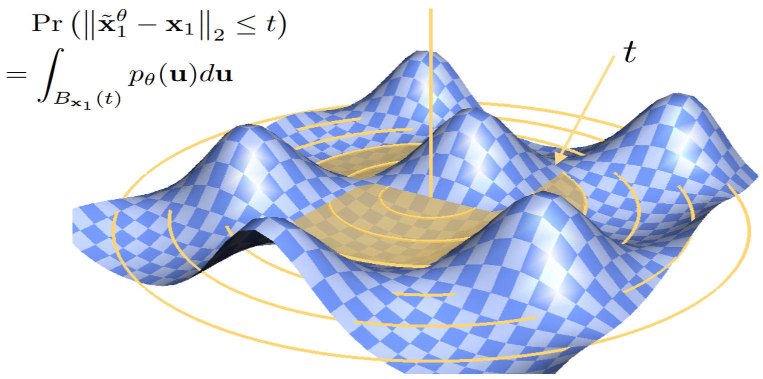

$\Pr\left(\|\tilde{\mathbf{x}}_1^\theta - \mathbf{x}_1\|_2 \leq t\right)$
$= \int_{B_{\mathbf{x}_1}(t)} p_\theta(\mathbf{u})d\mathbf{u}$
t

Figure 4.2: Probability density of the model P_θ.

In figure (4.3) we present the graph of the cumulative distribution function $Pr(|| \widetilde{x}_1^\theta - x_1 ||_2 \leq t)$ as a function of distance t. More specifically, the left graph represents the cumulative distribution function which is slowly increasing, indicating that the probability of the area around the data point x_1 is small. Finally, in the right graph, we notice that the cumulative distribution function increases rapidly, which means that the probability of the area around the data point x_1 is high.

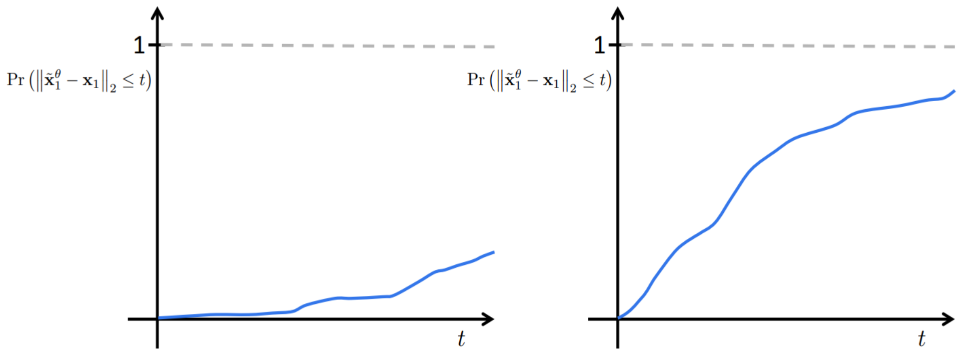


Figure 4.3: Cumulative distribution function as a function of distance.

The likelihood is maximized when the cumulative distribution function increases rapidly. To achieve this, we need to increase the area under the curve of the cumulative distribution function.

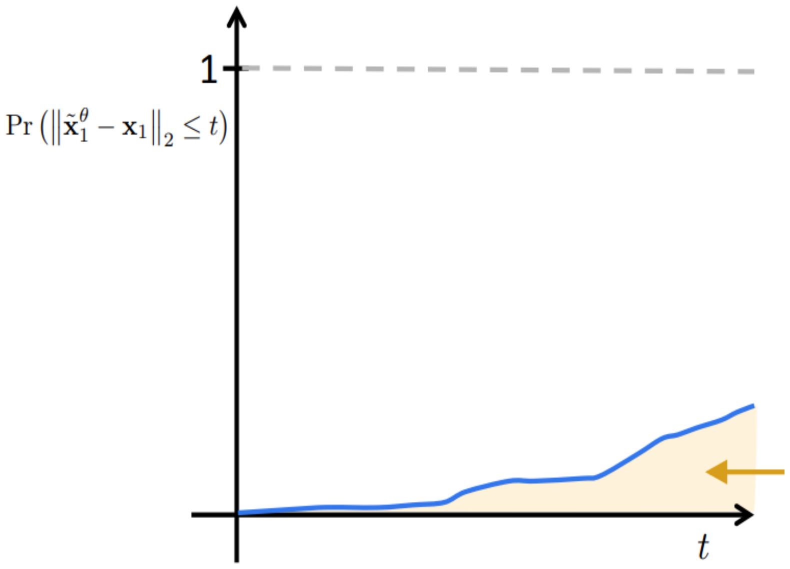

1
$\Pr\left(\left\|\tilde{\mathbf{x}}_1^{\theta} - \mathbf{x}_1\right\|_2 \leq t\right)$
t

Figure 4.4: Maximizing the cumulative distribution function (1st way).

Alternatively, we can reduce the area above the curve of the cumulative distribution function, because $1 - Pr(\| \widetilde{x}_1^{\theta} - x_1 \|_2 \leq t) = Pr(\| \widetilde{x}_1^{\theta} - x_1 \|_2 > t)$ holds. This area is equal to the expected value as $\int_0^{\infty} Pr(\| \widetilde{x}_1^{\theta} - x_1 \|_2 > t)dt = \mathbb{E}[\|\widetilde{x}_1^{\theta} - x_1\|_2]$, as for any non-negative random variable the following definition holds: $\mathbb{E}[x] = \int_0^{\infty} Pr(x > t)dt$.

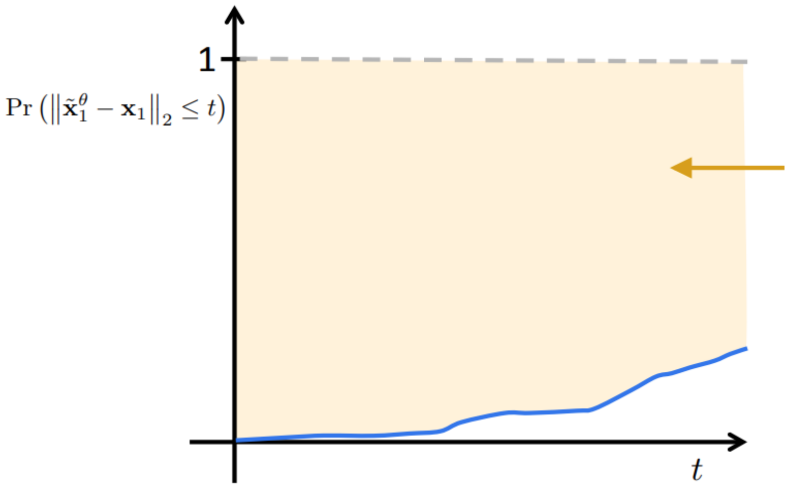

1
$\Pr\left(\left\|\tilde{\mathbf{x}}_1^{\theta} - \mathbf{x}_1\right\|_2 \leq t\right)$
t

Figure 4.5: Maximizing the cumulative distribution function (2nd way).

4.6 Implementing IMLE with a DNN

In order to implement the IMLE [3] method by the use of a DNN model, we are going to use the following framework.

- Initially, as an input to the DNN, we will define a random vector z of d dimension, sampled from the normal distribution $(z \sim N(0, I))$.
- We define the hidden neurons of the DNN according to the type of training examples. For image synthesis, for example, the optimal choice is to use a Deconvolutional DNN.
- Finally, we define the output level, depending on the type of examples we want to generate. We also make sure that the output layer of the network has the same dimension as the training examples.

The hidden layers of the DNN generative model define a function T_θ that maps the z random vector from the latent space to the y output space.

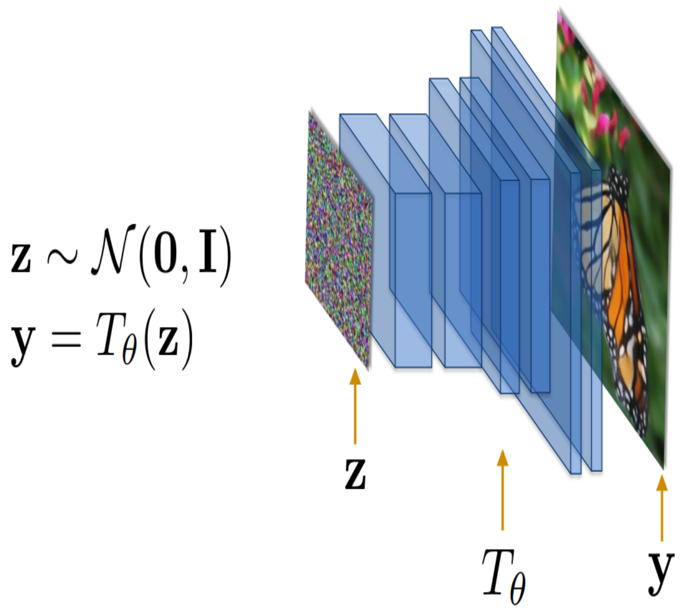


Figure 4.6: Generation of fake images by the use of IMLE [3].

Chapter 5

Deep Clustering Based on IMLE

5.1 Introduction

5.2 Cluster-Friendly Sampling Prior

5.3 Two-Stage Nearest Neighbor Search

5.4 The Encoder's Network

5.5 The Objective Function

5.6 The proposed method

5.1 Introduction

Generative models that DNNs implement, like GANs [2] and VAE [37], require a random noise vector z as an input to their latent space to generate samples. The z random noise vector is usually sampled by the normal distribution as $z \sim \mathcal{N}(0, I)$. The same is true for the IMLE [3] methodology discussed in Chapter 4. Although this procedure works fine for the generative process, it fails when used for clustering purposes at the DNNs' latent space. In figure 5.1, we present this problem in a two dimensional latent space example. The figure shows the input-output correspondence of a neural network that has learned to produce samples from five different clusters from the MNIST [58] corresponding the handwritten numbers 0, 1, 2, and 4. Each different color corresponds to a different cluster, dots with the blue color correspond to the zero cluster, dots with the pink color correspond to the one cluster etc.

In addition, another main disadvantage of this method is that producing a generated sample from a specific cluster is complicated. For example, let us suppose that we generate a batch of samples by the generator's model. In order to do that, we sample a batch of random vectors from the Gaussian distribution ($z \sim \mathcal{N}(0, I)$), and then we give them as input to the generative network. The problem emerges when verifying if the generated sample belongs to the cluster of samples we wished to generate. This procedure repeats until we successfully generate samples from the targeted cluster. If this is not true, then this procedure must be repeated. This problem exists because there is no prior knowledge of which areas of the latent space $\mathcal{Z}$ are responsible for creating samples from the respective cluster by the model.

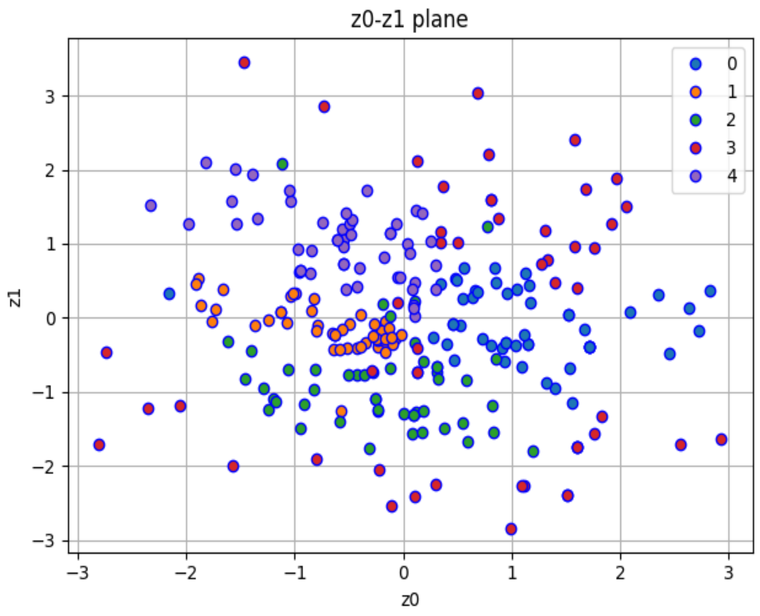


Figure 5.1: Input-output correspondence of a generative neural network. The 2-dimensional input is represented by z_0, z_1. The different colors represent the category of the generated sample produced by the neural network for the respective input (z_0, z_1).

In this chapter, we will present the essential algorithmic steps and modifications that took place in order to cluster the data in the latent space $\mathcal{Z}$ of the generator's DNN model by the use of IMLE [3]. First, we present a brief presentation of the main algorithmic ideas that we will extensively discuss in the next sections:

- A more cluster-friendly sampling prior.

- A new nearest neighbor searching algorithm, the two-stage nearest neighbor search.
- An additional DNN, the Encoder, which is responsible for clustering the data in the latent space $\mathcal{Z}$.
- An updated objective function that consists of clustering and generative (non-clustering) losses.

5.2 Cluster-Friendly Sampling Prior

Even though the vallina sampling prior cannot create discrete clusters in the latent space $\mathcal{Z}$, it appears that the model is learning to generate similar samples (of the same category) in nearby areas at the latent space $\mathcal{Z}$. However, the areas that correspond to each category are not distinct and as a result, they overlap to each other. This observation becomes clear in the figure 5.1. This means that the generator's model, implemented by a DNN, has an internal property of clustering the samples, meaning that similar samples are correspond to nearby areas of latent space. We will use this property of the generator's DNN to cluster data in the latent space $\mathcal{Z}$.

5.2.1 Mixture of Gaussians Distribution

In order to confirm whether the generator's model can cluster the data at the latent space, the first idea is to replace the sampling prior of the z random vector with a more cluster-friendly prior. More specifically:

- The sampling of z will not be performed from the Gaussian distribution $\mathcal{N}(0, I)$, but from a mixture of Gaussians distribution and every component$_i$ of this distribution will be denoted as follows $\mathcal{N}(\mu_i, \sigma^2 I)$. We chose σ^2 to be common for all the Gaussian components.
- The μ_i and σ^2 hyperparameters are selected appropriately, so that the samples from each component do not overlap with each other.
- The number of the Gaussian components must be set equal to the number of clusters which is specified by the user.

- Each Gaussian component has the same prior probability to be selected for sampling.

In figure 5.2 samples from a mixture of Gaussians distribution with four components. All Gaussians components have the same σ^2 which is equal to one. We selected the centers of the four Gaussians components as follows, $\mu_1 = (0, 0)$, $\mu_2 = (9, 0)$, $\mu_3 = (0, 9)$, and $\mu_4 = (9, 9)$. It is easy to identify which samples were produced by each Gaussian component.

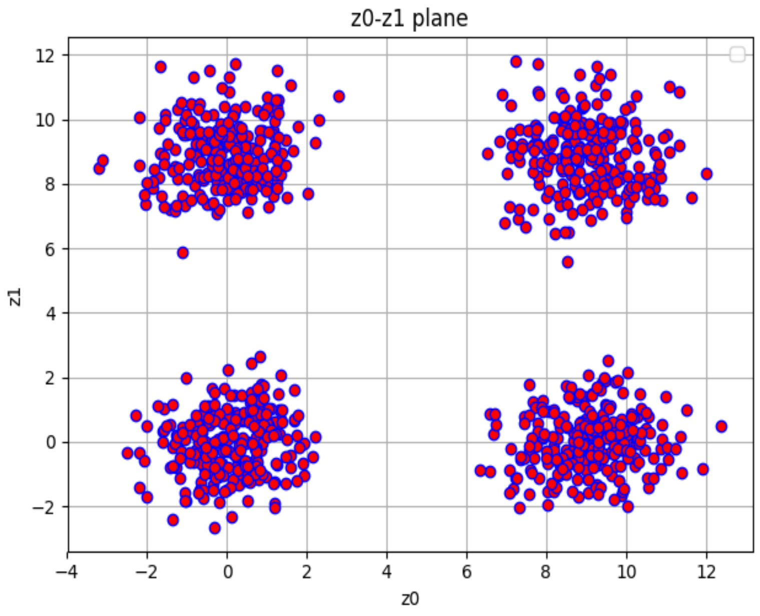


Figure 5.2: Samples generated by a a mixture of Gaussians distribution.

The following procedure is used to generate samples using the mixture of Gaussians distribution during the training phase of the generator's model. Suppose S denotes the total number of samples, and the number of Gaussian components is denoted by g. Algorithm 5.1 describes the data generation procedure corresponding to a mixture of Gaussians distribution:

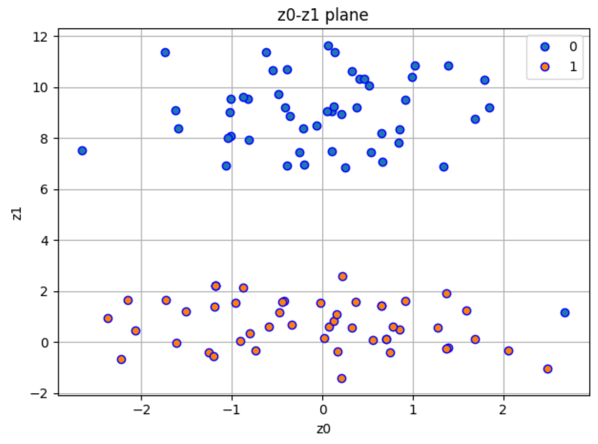
Algorithm 5.1 Generating samples through a mixture of Gaussians distribution.

Require: Sampling mechanism from a mixture Gaussians distribution and the generator's model T_θ.

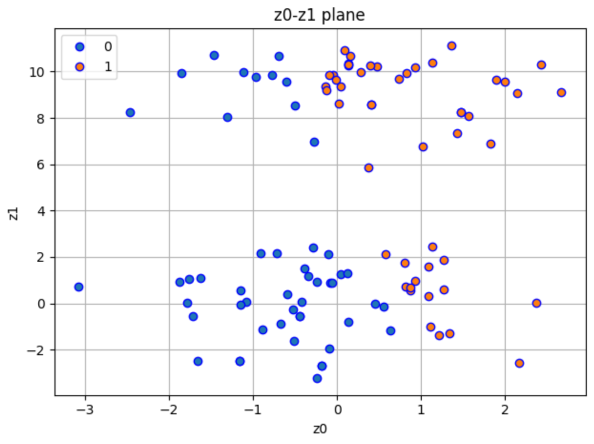
Ensure: Every Gaussian component has the same prior probability to be selected for sampling which is equal to $\frac{1}{g}$.

```
1: $t \leftarrow 0$
2: $samples \leftarrow \varnothing$
3: while $t < T$ do
4:     With probability equal to $\frac{1}{g}$, select one of $g$ Gaussian components. Suppose that $g_i$ is selected with $N(\mu_i, \sigma^2 I)$.
5:     Sample a random vector $z_i$ for the selected Gaussian component $g_i$ denoted as $z_i \sim N(\mu_i, \sigma^2 I)$.
6:     $samples \leftarrow samples \cup \{T_\theta(z_i)\}$
7:     $t \leftarrow t + 1$
8: end while
9: Return the generated $samples$
```

Figure 5.3 presents the results that were generated while using algorithm 5.1 for sampling the z random vectors as an input to the generator's model. The training data belong to two clusters, and they are selected from the MNIST [58] (28x28 handwritten images). They consist of the handwritten numbers zero and one. We selected the z inputs to be two-dimensional for graphical representation purposes. Both Gaussian components have the same σ^2 which is equal to one but different μ. More specifically, for the first Gaussian we set $\mu_0 = (0, 0)$ and for the second $\mu_1 = (0, 9)$. Figure 5.3a shows that with this method, the model can cluster the training data in the z latent space (input layer). However, it is not certain that the clustering will always be successful, as shown in figure 5.3b. This experiment shows that the generator's DNN trained by the IMLE method can cluster the data successfully in the latent space by using a more cluster-friendly sampling prior.



(a)



(b)

Figure 5.3: 5.3a Training results using a mixture of Gaussians distribution resulting in successful clustering. 5.3b Training results using a mixture of Gaussians distribution resulting in unsuccessful clustering.

5.2.2 Mixture of Discrete and Continuous Latent Variables

Even though the previous sampling method is suitable for clustering in the latent space $\mathcal{Z}$, its main disadvantage is the considerable number of hyperparameters. More precisely, for every Gaussian component, it is required to set its μ and σ^2 and afterward double-check that the samples from each Gaussian component do not overlap with each other. In order to overcome this problem, we utilized the ClusterGans' sampling prior [1] we described in section 3.3.1 which is a suitable and clever method to avoid this kind of problem. More specifically, as input to the generator, we utilize a mixture of discrete and continuous latent variables to create a non-smooth geometry in the latent space. This is a useful property so that the constructed clusters in the latent space will be strictly separated from each other. To do that, we sample from a prior that consists of normal random variables cascaded with one-hot encoded vectors. To be more precise the latent variable is defined as $z = (z_n, z_c)$ where $z_n \sim \mathcal{N}(0, \sigma^2 I)$ and the latent vector z_c is sampled by an one-hot distribution of K elements (K is the number of clusters). We select σ equal to a small value like $\sigma = 0.10$ so that for each dimension of the normal latent variables, $z_{n,j} \in (-0.6, 0.6) << 1.0\ \forall j$ holds with high probability. The choice of small value for standard deviation σ is crucial so that the clusters constructed in the latent space $\mathcal{Z}$ of the generator network will not overlap but remain separated.

5.3 Two-Stage Nearest Neighbor Search

We increased the number of Gaussian components, utilizing a mixture of discrete and continuous latent variables, in order to have at least one Gaussian component for each different data cluster. This way, the generator has the opportunity to assert in each Gaussian component samples from a distinct cluster. However, we still face the same problem in the clustering procedure. The generators' model has much flexibility (implemented by a DNN), and at the learning phase, one Gaussian component may attract samples belonging to different clusters, as we already presented in figure 5.3b.

The following major algorithmic change we propose concerns how the IMLE selects the nearest generated sample for each real data example. It should be noted that the sampling prior to our method is a mixture of discrete and continuous latent variable (subsection 5.2.2). We recall that the IMLE algorithm searches for the nearest generated sample to each real data example in the entire set of generated samples. This works perfectly for the data generation procedure. However, it can create major problems to clustering in latent space $\mathcal{Z}$ because each Gaussian component can attract more than one cluster from the dataset, counterproductively to clustering.

In order to overcome this problem, we have modified the nearest neighbor search into a two-stage procedure. Let us suppose that we generated samples (a mixture of discrete and continuous latent variables) from each Gaussian component. Then we group the generated samples based on the Gaussian component that are sampled from. By the use of each samples' grouping, we compute the "synthetic" centroids, which means that the samples of each Gaussian component generate a synthetic centroid. This way, the number of synthetic centroids is equal to the number of Gaussian components. Thus, in the first stage of the nearest neighbor search, we find the nearest synthetic centroid for each real data example. In the second stage, for every data example, we execute a nearest neighbor search on the set of generated samples, which computed by the nearest synthetic centroid. More specifically, the detailed steps of this procedure are the following:

- **Step 1**: Generation of synthetic samples using the generator network during the training procedure using a mixture of discrete and continuous latent variables as described subsection 5.2.2.
- **Step 2**: Computation of the centroid of each Gaussian component. The centroid is computed by the mean value of the synthetic samples produced by that

particular Gaussian component.

- **Step 3**: In this step we execute the first stage of the two-stage nearest neighbor search. For each data example, search for the nearest synthetic centroid.
- **Step 3**: In this step we execute the second stage of the two-stage nearest neighbor search. For every data example, we execute a nearest neighbor search on the set of generated samples, which computed by the nearest synthetic centroid to it.

Algorithm 5.2 describes the two-step nearest neighbor search.

Algorithm 5.2 The two-step nearest neighbor search algorithm.

```
Require: Mixture of discrete and continuous latent variable sampling mechanism and
    the generator's model $T_\theta$.
1: Generation of $S$ synthetic samples drawn from $g$ Gaussian components using the
    mixture of discrete and continuous latent variable sampling mechanism.
2: Computation of the $C$ centroids of the Gaussians components.
3: For each data example $x_i$ find the nearest centroid $C_j$.
4: For each data example $x_i$ find the nearest synthetic sample $s_k$ under the constraint
    that $s_k \cap C_l = \emptyset, \forall l \neq j$.
5: Return the nearest synthetic samples for each data example.
```

5.3.1 Training the Generator

In order to train the generator network and cluster the training data, we utilize the centroids that are computed from the synthetic samples as described in algorithm 5.2. The main training steps to achieve that are presented below:

- Compute the cluster synthetic centroids and find the nearest sample to each data example using the training algorithm 5.2.
- Assign each training data point to the cluster whose synthetic centroid is nearest to it.
- Update the generator network parameters using the IMLE update procedure.

It is worth mentioning that this method is iterative, and the clusters tend to improve as the training proceeds. Furthermore, it is worth mentioning that as the training epochs are executed, the clustering error is minimized. The complete training algorithm, which makes use of the IMLE model and the two-stage nearest neighbor search, is presented in Algorithm 5.3.

Algorithm 5.3 Clustering via the IMLE model and two-stage nearest neighbor search.

```
Require: Two-step nearest neighbor search algorithm 5.2.
1: $t \leftarrow 1$
2: while $t < T$ do
3:     Computation of the synthetic centroids $C$ and of the set of nearest synthetic samples $\widetilde{S}$ to data examples $X$ using the two-stage nearest neighbor search algorithm 5.2.
4:     Creation of data clusters by assigning each data example $x_i$ to the cluster whose synthetic centroid $c_j$ is nearest to it.
5:     Update the generator's parameters: $\theta \leftarrow \theta - \eta \nabla_\theta (\frac{n}{|\widetilde{S}|} ||\widetilde{S} - X||_2^2)$, where $\widetilde{S}, X$ matrices.
6:     $t \leftarrow t + 1$
7: end while
8: Return the data clusters.
```

5.3.2 Similarities with k-means

At this point, it is evident that the algorithm 5.3 has many similarities with the k-means algorithm. Both algorithms are iterative, and they tend to get better clustering results by each iteration. In the table 5.1 we present some of the main similarities of those two clustering procedures. It could be argued that the proposed approach can be considered as an enhancement of k-means that is able to accommodate more complex data.

Table 5.1: Similarities of the k-means algorithm and the modified IMLE algorithm.

	k-means	Modified IMLE
Iterative Algorithm	True	True
Random Initialization	Centroids	Generator's parameters
Clustering Error	Direct Minimization	Implicit Minimization
Updating the Cluster Centroids	Direct Centroids Updates	Updating the Generator's Parameters
Operates on the Original Data Space	True	True

5.4 The Encoder's Network

Even though we can achieve clustering already with the proposed algorithm 5.3, this algorithm operates in the original data space without fully utilizing the transformational abilities the DNNs can provide. Therefore, in order to make use of useful deep features and the representational power of DNNs, we have modified the vanilla architecture of the IMLE [3] by adding a second network, the Encoder. The Encoder's network is used similarly as in the ClusterGan algorithm [1], which has been described in subsection 3.3.2, in order to assist in clustering.

Thus the proposed method contains two networks, the Generator, and the Encoder. More specifically, the Generator is responsible for generating synthetic samples based on the training data. As input, the Generator expects some random vector which is sampled by a mixture of discrete and continuous distribution (subsection 5.2.2). The Generator is a transformation from the latent space $\mathcal{Z}$ to data space X, $\mathcal{G} : \mathcal{Z} \to X$. The Encoder takes as input a generated sample X_g and its primary functionality is to map it from data space X back to latent space $\mathcal{Z}$, $\mathcal{E} : X \to \mathcal{Z}$. Thus, using the Encoder is an efficient way to train in an unsupervised classification manner to cluster the input data. For the purpose of better understanding the new IMLE Generators-Encoder methodology, in figure 5.4 we present its architecture and its basic operation. As an input the generator $\mathcal{G}$ takes z which is a mixture of discrete and continuous random variable, $z = (z_n, z_c)$ and creates synthetic samples X_g. The Encoder receives the synthetic sample X_g and inverse-maps it back to the $\mathcal{Z}$ latent space, producing a latent vector $\hat{z}$.

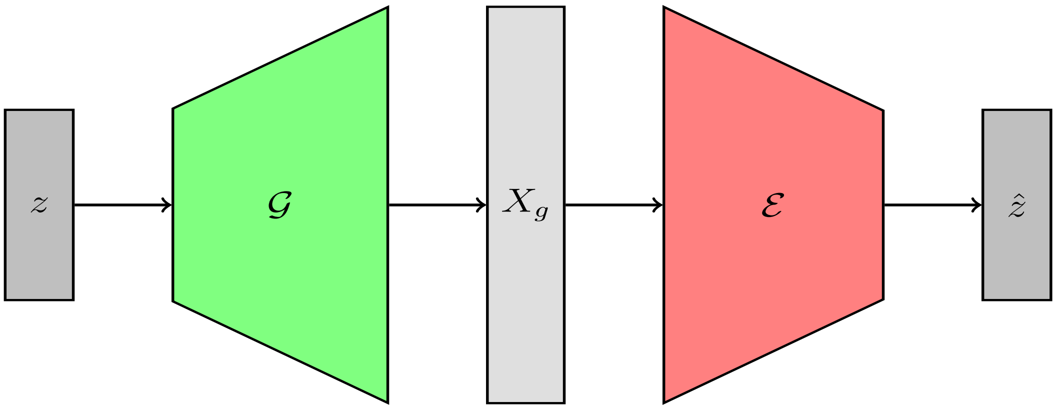


Figure 5.4: IMLE Generator-Encoder Architecture.

5.5 The Objective Function

The objective function used to train the network contains a part from the generative process and a part from the clustering procedure. As we already analyzed in chapter 2, the deep clustering objective function consists of a clustering and a non-clustering loss. *It is should be noted that, with the symbol $\widetilde{x}_{\theta_G}^{(i)} = \mathcal{G}_{\theta_G}(z^{(i)})$ we denote the generated sample which is closest to the data example x_i as found by the two-stage nearest neighbor search.*

The part of the objective function that concerns the generative process (non-clustering loss), is the IMLE [3] error which is equal to $\sum_{i=1}^{n} ||\mathcal{G}_{\theta_G}(z^{(i)}) - x^{(i)}||_2^2 = \sum_{i=1}^{n} ||\widetilde{x}_{\theta_G}^{(i)} - x^{(i)}||_2^2$. Furthermore, we have added a generative loss term that makes use of the Encoder's extracted deep features. More specifically, we input each real data and each closest generated sample found by the two-stage nearest neighbor into the Encoder's network. Then, we extract the deep feature representation that the Encoder computed at its hidden layers. Using the deep feature representation of each data and of its closest sample we construct the following loss term, $\sum_{i=1}^{n} ||\mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z^{(i)}))_f - \mathcal{E}_{\theta_E}(x^{(i)})_f||_2^2 = \sum_{i=1}^{n} ||\mathcal{E}_{\theta_E}(\widetilde{x}_{\theta_G}^{(i)})_f - \mathcal{E}_{\theta_E}(x^{(i)})_f||_2^2$. The reason for that additional generational term is that in order to classify complex data, it is necessary to extract complex features. This is a procedure usually done automatically by DNN architectures and yield satisfactory results. Because the Encoder's model is trained in an unsupervised classification manner, we make of the deep representation features constructed through its training. It is worth mentioning that the Generator outputs generated samples that resemble random noise at the beginning of its training. These kinds of generated samples seem reasonable because the Generator is untrained at this stage of the training. These two error terms aim to improve these outputs by implicitly maximizing the model's likelihood by using the original image space and the embedded one that is constructed by the Encoder.

The rest part of the objective function terms, the clustering loss terms, aims to train the Encoder's DNN to inverse-map the generated samples X_g back to the latent space, $\hat{z} = \mathcal{E}_{\theta_E}(X_g)$ which equals to $\hat{z} = \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z))$. In this way, the trained Encoder is clustering the data based on the z_c part, meaning the discrete part of the random variable z, which encodes the cluster or equivalently the Gaussian component that created the sample. Finally we added two additional terms to the objective function 5.1. The first term, $\sum_{i=1}^{n} ||z_n^{(i)} - \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z_n^{(i)}))||_2^2$, is responsible for inverse-mapping the z_n part of the z random noise which is sampled from the Gaussian distribution $z_n \sim \mathcal{N}(0, \sigma^2 I_{d_n})$, and acts as a regularization of the continuous portion z_n of the latent space. This ensures the reversibility of the continuous portion z_n of the latent space given a sample $\mathcal{E} : \mathcal{G}(z_n) \rightarrow z_n$. The second term, $\sum_{i=1}^{n} \mathcal{H}(z_c^{(i)}, \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z_c^{(i)})))$, is responsible for inverse-mapping the z_c part of the z random noise which is sampled by one-hot distribution, where $\mathcal{H}(.,.)$ is the cross-entropy loss, which is defined as $\mathcal{H}(X) = -\sum_{x} p(x)\log(p(x))$. Finally, the last term is responsible for training the Encoder to cluster the data, via an unsupervised classification procedure meaning that for all the mappings of points that belong in the same class in $\mathcal{X}$ space to have the same one-hot encoding when embedded in $\mathcal{Z}$ space. This regularization is the most critical part for a successful data clustering in the latent space $\mathcal{Z}$.

The final objective function goes beyond simple distances in data space. It can capture complex and perceptually important properties of data through the mixture of losses in data and feature spaces [57]. The final objective function 5.1 is presented below:

$$\begin{aligned} J(\theta_G, \theta_E) = \alpha \sum_{i=1}^{n} ||\mathcal{G}_{\theta_G}(z^{(i)}) - x^{(i)}||_2^2 + \alpha_f \sum_{i=1}^{n} ||\mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z^{(i)}))_f - \mathcal{E}_{\theta_E}(x^{(i)})_f||_2^2 \\ +\beta_n \sum_{i=1}^{n} ||z_n^{(i)} - \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z_n^{(i)}))||_2^2 + \beta_c \sum_{i=1}^{n} \mathcal{H}(z_c^{(i)}, \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z_c^{(i)}))) \end{aligned} \tag{5.1}$$

where the $\alpha, \alpha_f, \beta_n$, and β_c are regularization hyperparameters.

5.6 The proposed method

It is time to put everything together and combine all the necessary algorithmic steps to develop the final deep clustering method, which is based on IMLE. It should be noted that at this point, we already defined the Generator's and Encoder's DNNs model, and we obtained the training dataset. The steps of the complete training algorithm are presented below:

- Create a batch of generated samples x_g by Generator's model, using as input a mixture of discrete and continuous latent variables $z = (z_n, z_c)$.
- Input all the generated samples x_g to the Encoder's model and compute as output we have the inverse-mapping of the generated samples $\hat{z}_g$.
- For each data example x_r^i search the nearest generated sample x_g^i with the use of the two stage nearest neighbor algorithm 5.3.
- For each data example x_r^i and its nearest sample x_g^i extract their deep feature representation, by the use of the Encoder's network.
- Compute the gradients of the objective function 5.1 using of Adam optimizer [16] (or any other gradient based learning rule), and back-propagate them in order to minimize the objective function.

After the end of the learning process, we have obtained the trained Generator's model, which can generate samples, and the trained encoder's model that clusters the input data. The complete training algorithm of the deep clustering method is presented in Algorithm 5.4.

Algorithm 5.4 Deep Clustering Based on IMLE

```
Require: Setting the hyperparameters $\alpha, \alpha_f, \beta_n, \beta_c$.
 1: $t \leftarrow 1$
 2: while $t \leq T$ do
 3:     Sample minibatch of $m$ noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from mixture of discrete and continuous noise prior $\mathbb{P}_z$, where $z = (z_n, z_c)$.
 4:     Sample minibatch of $m$ generated samples $\{x_g^{(1)}, ..., x_g^{(m)}\}$ from the Generator's model $\mathcal{G}_{\theta_G}(z)$.
 5:     Inverse map the generated samples $X_g$ using the Encoder model $\mathcal{E}_{\theta_E}(X_g)$ to obtain $\{\hat{z}^{(1)}, ..., \hat{z}^{(m)}\}$.
 6:     For the data examples $X_r$ find the nearest samples $X_g^{ns}$ with the two-stage nearest neighbor search.
 7:     Update $z$ to only contain the priors of the nearest samples $X_g^{ns}$.
 8:     For each data example $x_r^{(i)}$ and nearest sample $x_g^{ns_i}$ extract their deep feature representation $x_f^{(i)}$ and $\mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z^{(i)}))_f$ respectively, by the use of the Encoder's network.
 9:     Update the generator by descending its stochastic gradient:
            $$\nabla_{\theta_\mathcal{G}} \frac{1}{m} \Bigg( \alpha \sum_{i=1}^{n} ||x^{(i)} - \mathcal{G}_{\theta_G}(z^{(i)})||_2^2 + \alpha_f \sum_{i=1}^{n} ||\mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z^{(i)}))_f - \mathcal{E}_{\theta_E}(x^{(i)})_f||_2^2 + \beta_n \sum_{i=1}^{m} ||z_n^{(i)} - \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z_n^{(i)}))||_2^2 + \beta_c \sum_{i=1}^{m} \mathcal{H}(z_c^{(i)}, \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z_c^{(i)}))) \Bigg) \quad (5.2)$$
10:     Update the encoder by descending its stochastic gradient:
            $$\nabla_{\theta_\mathcal{E}} \frac{1}{m} \Bigg( \alpha_f \sum_{i=1}^{n} ||\mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z^{(i)}))_f - \mathcal{E}_{\theta_E}(x^{(i)})_f||_2^2 + \beta_n \sum_{i=1}^{m} ||z_n^{(i)} - \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z_n^{(i)}))||_2^2 + \beta_c \sum_{i=1}^{m} \mathcal{H}(z_c^{(i)}, \mathcal{E}_{\theta_E}(\mathcal{G}_{\theta_G}(z_c^{(i)}))) \Bigg) \quad (5.3)$$
11:     $t \leftarrow t + 1$
12: end while
13: Return the Generator and the Encoder.
```

The gradient-based updates can use any standard gradient-based learning rule. The most popular choice is the Adam optimizer [16].

Chapter 6

Experiments

6.1 Datasets

6.2 Hyperparameter and Architecture Details

6.3 Experiments

6.4 Image Generation

6.5 Dimensionality Reduction

6.6 Training Procedure

6.1 Datasets

In our experiments, we included different datasets with several data complexities and data types. We used famous datasets that are commonly utilized to benchmark various image processing systems and machine learning algorithms. The datasets used are the following:

- The MNIST [58] is a dataset consisting 60,000 samples for training and 10,000 samples for testing. Each sample is a 28×28 grayscale image containing handwritten digits associated with a label from ten classes. We also used a simpler version of the MNIST dataset that contains four of the total ten classes. The selected classes are Zero, One, Two, and Three.

- The Fashion-MNIST [60] is a dataset consisting 60,000 samples for training and 10,000 samples for testing. Each sample is a 28×28 grayscale image containing a fashion product associated with a label from ten classes. We can say that Fashion-MNIST is a more complicated version of the MNIST dataset. We also merged some categories which were similar to form a separate 5-class dataset. The five groups were as follows : {Tshirt/Top, Dress}, {Trouser}, {Pullover, Coat, Shirt}, {Bag}, {Sandal, Sneaker, Ankle Boot}.

- The Pendigits [61] is a dataset consisting of 10992 writing samples from 44 different writers, in total 10992 written samples. Each sample is a 1×16 vector, containing pixel coordinates associated with a label from ten classes. We also defied two additional datasets. The first one contains only the odd numbers, and the second only the even numbers of the Pendigits dataset.

- The Synthetic dataset is generated from a mixture of Gaussians with four components. Each generated sample is 1×2 vector (2 dimensional). We generated 100 points from each Gaussian component so that the total number of samples equals 400.

It is worth mentioning that we did not use the complete datasets, but we sampled a small subset randomly from each of them to create a smaller version of each dataset. We ensured that every data class had the same number of training data points included. The subsets that we created from each dataset are the following:

- MNIST [58]: which 50 data points sampled for each of the total ten classes (in total 500 training data).

- MNIST, 4 Classes [58]: which 50 data points sampled for each of the four selected classes (in total 250 training data).

- Fashion-MNIST [60]: which 50 data points sampled for each of the total ten classes (in total 500 training data).

- Fashion-MNIST, 5 Classes [60]: which 100 data points sampled for each of the total five classes (in total 500 training data).

- Pendigits [61]: which 200 data points sampled for each of the total ten classes (in total 2000 training data).

- Pendigits, Odd Numbers [61]: which 400 data points sampled for each of the total five classes (in total 2000 training data).

- Pendigits, Even Numbers [61]: which 400 data points was sampled for each of the total five classes (in total 2000 training data).

6.2 Hyperparameter and Architecture Details

For the training of the ClusterGan [1] and the IMLE based model, we used the Adam optimizer [16]. We selected as Adams' [16] hyperparameters the following, $\eta = 1e - 4$, $\beta_1 = 0.5$, $\beta_2 = 0.99$ for all datasets. In every experiment, we execute 500 training epochs with a batch size equal to 64. It is worth mentioning that for the better comparison of the ClusterGan [1] with the IMLE based model, we chose both methods to share common architectures for the Generator and Encoder networks. For the sake of simplicity, we will present their architectures for each dataset in the same table. Of course, the Discriminator network concerns only the ClusterGan's [1] architecture. The dimension of z_c is the same as the number of data clusters in each dataset. The networks use ReLU activations, Leaky ReLU activations ($leak = 0.2$) and Batch Normalization (BN). Next, we present the architectural details for training the deep clustering methods to each different dataset.

Synthetic Data. We used $z_n = 10$ and $z_c = 4$ the same number as the number of classes in this dataset.

Table 6.1: The Generator's, Encoder's and Discriminator's architecture for the Synthetic dataset.

Generator	Encoder	Discriminator
$z = (z_n, z_c) \in \mathbb{R}^{14}$	Input $X \in \mathbb{R}^2$	Input $X \in \mathbb{R}^2$
FC 256, LReLU, BN	FC 256, LReLU, BN	FC 256, LReLU, BN
FC 256, LReLU, BN	FC 256, LReLU, BN	FC 256, LReLU, BN
FC 2, Sigmoid	FC 14, linear for $\hat{z}_n$ and Softmax for $\hat{z}_c$	FC 1, Sigmoid

Pendigits Data, Odd Numbers. We used $z_n = 10$ and $z_c = 5$ the same number as the number of classes in this dataset.

Table 6.2: The Generator's, Encoder's and Discriminator's architecture for the Pendigits dataset with only odd numbers.

Generator	Encoder	Discriminator
$z = (z_n, z_c) \in \mathbb{R}^{15}$	Input $X \in \mathbb{R}^{16}$	Input $X \in \mathbb{R}^{16}$
FC 256, LReLU, BN	FC 256, LReLU, BN	FC 256, LReLU, BN
FC 256, LReLU, BN	FC 256, LReLU, BN	FC 256, LReLU, BN
FC 16, Sigmoid	FC 15, linear for $\hat{z}_n$ and Softmax for $\hat{z}_c$	FC 1, Sigmoid

Pendigits Data, Even Numbers. We used $z_n = 10$ and $z_c = 5$ the same number as the number of classes in this dataset.

Table 6.3: The Generator's, Encoder's and Discriminator's architecture for the Pendigits dataset with only even numbers.

Generator	Encoder	Discriminator
$z = (z_n, z_c) \in \mathbb{R}^{15}$	Input $X \in \mathbb{R}^{16}$	Input $X \in \mathbb{R}^{16}$
FC 256, LReLU, BN	FC 256, LReLU, BN	FC 256, LReLU, BN
FC 256, LReLU, BN	FC 256, LReLU, BN	FC 256, LReLU, BN
FC 16, Sigmoid	FC 15, linear for $\hat{z}_n$ and Softmax for $\hat{z}_c$	FC 1, Sigmoid

Pendigits Data. We used $z_n = 10$ and $z_c = 10$ the same number as the number of classes in this dataset.

Table 6.4: The Generator's, Encoder's and Discriminator's architecture for the Pendigits dataset.

Generator	Encoder	Discriminator
$z = (z_n, z_c) \in \mathbb{R}^{20}$	Input $X \in \mathbb{R}^{16}$	Input $X \in \mathbb{R}^{16}$
FC 256, LReLU, BN	FC 256, LReLU, BN	FC 256, LReLU, BN
FC 256, LReLU, BN	FC 256, LReLU, BN	FC 256, LReLU, BN
FC 16, Sigmoid	FC 20, linear for $\hat{z}_n$ and Softmax for $\hat{z}_c$	FC 1, Sigmoid

MNIST and Fashion-MNIST. We used $z_n = 10$ and $z_c = 10$ the same number as the number of classes in these datasets.

Table 6.5: The Generator's, Encoder's and Discriminator's architecture for the MNIST and Fashion-MNIST datasets.

Generator	Encoder	Discriminator
$z = (z_n, z_c) \in \mathbb{R}^{20}$	Input $X \in \mathbb{R}^{28\times 28}$	Input $X \in \mathbb{R}^{28\times 28}$
512, 1×1 upconv., 1 stride, ReLU, BN	64, 4×4 upconv., 2 stride, LReLU	64, 4×4 conv., 2 stride, LReLU
512, 1×1 upconv., 1 stride, ReLU, BN	128, 4×4 upconv., 2 stride, LReLU	128, 4×4 conv., 2 stride, LReLU
256, 7×7 upconv., 1 stride, ReLU, BN	FC 1024, LReLU	FC 1024, LReLU
128, 4×4 upconv., 2 stride, 1 padding, ReLU, BN	FC 20, linear for $\hat{z}_n$ and Softmax for $\hat{z}_c$	FC 1, Sigmoid
1, 4×4 upconv., 2 stride, 1 padding, Sigmoid	-	-

MNIST, 4 Classes. We used $z_n = 10$ and $z_c = 4$ the same number as the number of classes in these datasets. We choose the numbers zero, one, two and three.

Table 6.6: The Generator's, Encoder's and Discriminator's architecture for the MNIST datasets with four classes.

Generator	Encoder	Discriminator
$z = (z_n, z_c) \in \mathbb{R}^{14}$	Input $X \in \mathbb{R}^{28\times28}$	Input $X \in \mathbb{R}^{28\times28}$
512, 1×1 upconv., 1 stride, ReLU, BN	64, 4×4 upconv., 2 stride, LReLU	64, 4×4 conv., 2 stride, LReLU
512, 1×1 upconv., 1 stride, ReLU, BN	128, 4×4 upconv., 2 stride, LReLU	128, 4×4 conv., 2 stride, LReLU
256, 7×7 upconv., 1 stride, ReLU, BN	FC 1024, LReLU	FC 1024, LReLU
128, 4×4 upconv., 2 stride, 1 padding, ReLU, BN	FC 14, linear for $\hat{z}_n$ and Softmax for $\hat{z}_c$	FC 1, Sigmoid
1, 4×4 upconv., 2 stride, 1 padding, Sigmoid	-	-

Fashion-MNIST, 5 Classes. We used $z_n = 10$ and $z_c = 5$ the same number as the number of classes in these datasets. We choose the numbers zero, one, two and three.

Table 6.7: The Generator's, Encoder's and Discriminator's architecture for the fashion-MNIST datasets with five classes.

Generator	Encoder	Discriminator
$z = (z_n, z_c) \in \mathbb{R}^{15}$	Input $X \in \mathbb{R}^{28\times28}$	Input $X \in \mathbb{R}^{28\times28}$
512, 1×1 upconv., 1 stride, ReLU, BN	64, 4×4 upconv., 2 stride, LReLU	64, 4×4 conv., 2 stride, LReLU
512, 1×1 upconv., 1 stride, ReLU, BN	128, 4×4 upconv., 2 stride, LReLU	128, 4×4 conv., 2 stride, LReLU
256, 7×7 upconv., 1 stride, ReLU, BN	FC 1024, LReLU	FC 1024, LReLU
128, 4×4 upconv., 2 stride, 1 padding, ReLU, BN	FC 15, linear for $\hat{z}_n$ and Softmax for $\hat{z}_c$	FC 1, Sigmoid
1, 4×4 upconv., 2 stride, 1 padding, Sigmoid	-	-

6.3 Experiments

In this section, we will present the experiments that took place. First, it is necessary to mention that, Deep Clustering methodologies rely heavily on the random initialization of the DNNs' parameters, as with every other DL method. Therefore, it was necessary to perform every experiment at least 30 times. Afterward, we collected the experimental results and applied statistical analysis to obtain conclusions. Using this methodology, we can avoid edge case scenarios to the experimental results caused by bad initializations.

We executed a variety of experiments with different algorithmic setups, and in table 6.8 we explain the different algorithm choices we made to execute the following experiments. Throughout the experiments the ClusterGans' hyperparameters where set to default values which are $\beta_n = 10$ and $\beta_c = 10$. Additionally, we present the hyperparameters for each algorithmic variation of the methods based on IMLE [3] in table 6.9. We chose β_n to equal zero in the IMLE based methods because it did not make any significant difference to the clustering results.

Table 6.8: Presentation of different algorithmic variations used.

Algorithm	Prior Distribution	Clustering Method	N.N. search	Encoder	Deep Feautures
Simple IMLE	Mixture of Gaussian	Samples' Centroid	Simple	No	No
IMLE 2-s N.N	Mixture of Gaussian	Samples' Centroid	Two-Stage	No	No
IMLE-Encoder	Mixture of districrete and continuous	Encoder	Two-Stage	Yes	No
IMLE-Encoder Centroid	Mixture of districrete and continuous	Samples' Centroid	Two-Stage	Yes	No
Final IMLE-Encoder	Mixture of districrete and continuous	Encoder	Two-Stage	Yes	Yes
Final IMLE-Encoder Centroid	Mixture of districrete and continuous	Samples' Centroid	Two-Stage	Yes	Yes

Table 6.9: Presentation of the hyperparameters selection for each algorithm.

Algorithm	α	α_f	β_c	β_n
Simple IMLE	1.0	0.0	0.5	0.0
IMLE 2-s N.N.	1.0	0.0	0.5	0.0
IMLE-Encoder	1.0	0.0	0.5	0.0
IMLE-Encoder Centroid	1.0	0.0	0.5	0.0
Final IMLE-Encoder	1.0	0.5	0.5	0.0
Final IMLE-Encoder Centroid	1.0	0.5	0.5	0.0

In tables 6.10, 6.11 we compare the clustering algorithms based on the accuracy (ACC) and normalized mutual information (NMI) metrics. Based on these two tables, some empirical conclusions can be drawn when comparing the Deep Clustering method based on IMLE to ClusterGan. More specifically, the Deep Clustering method based on IMLE and its variations performs better at most datasets than ClusterGan. More specifically, the IMLE based method outperforms ClusterGan at the Synthetic, the MNIST (with ten and with four classes), the Fashion-MNIST (with ten and with five classes). The two algorithms had similar results in the Pendigits dataset and in the Pendigits dataset with only the odd digits. The ClusterGan was able to yield better results only at the Pendigits dataset with the even digits. Additionally, in Tables 6.12 to 6.27, we present more detailed statistical information about the algorithms that are based on the IMLE that had the most promising experimental results for each dataset separately.

Table 6.10: Comparison of clustering metrics across datasets.

Dataset	Algorithm	ACC	NMI
Synthetic	Simple IMLE	0.4	0.1
	IMLE 2-s N.N	**1**	**1**
	IMLE-Encoder	**1**	**1**
	IMLE-Encoder Centroid	**1**	**1**
	F. IMLE-Encoder	**1**	**1**
	F. IMLE-Encoder Centroid	**1**	**1**
	ClusterGAN	0.96	0.91
MNIST	Simple IMLE	0.44	0.37
	IMLE 2-s N.N	0.5	**0.44**
	IMLE-Encoder	**0.51**	**0.44**
	IMLE-Encoder Centroid	**0.51**	0.43
	F. IMLE-Encoder	**0.51**	**0.44**
	F. IMLE-Encoder Centroid	**0.51**	**0.44**
	ClusterGAN	0.25	0.12
MNIST 4 Classes	Simple IMLE	0.6	0.34
	IMLE 2-s N.N	**0.84**	**0.64**
	IMLE-Encoder	0.81	0.61
	IMLE-Encoder Centroid	0.81	0.61
	F. IMLE-Encoder	0.81	0.61
	F. IMLE-Encoder Centroid	0.8	0.61
	ClusterGAN	0.53	0.25
Fashion-MNIST	Simple IMLE	0.52	0.49
	IMLE 2-s N.N	0.56	0.52
	IMLE-Encoder	**0.57**	**0.54**
	IMLE-Encoder Centroid	**0.57**	**0.54**
	F. IMLE-Encoder	0.55	0.51
	F. IMLE-Encoder Centroid	0.56	0.52
	ClusterGAN	0.27	0.16

Table 6.11: Comparison of clustering metrics across datasets.

Dataset	Algorithm	ACC	NMI
Fashion-MNIST 5 Classes	Simple IMLE	0.6	0.38
	IMLE 2-s N.N	0.66	0.48
	IMLE-Encoder	**0.7**	**0.53**
	IMLE-Encoder Centroid	0.68	0.51
	F. IMLE-Encoder	0.66	0.49
	F. IMLE-Encoder Centroid	0.67	0.49
	ClusterGAN	0.44	0.13
Pendigits	Simple IMLE	0.66	0.65
	IMLE 2-s N.N	0.7	0.66
	IMLE-Encoder	**0.71**	**0.68**
	IMLE-Encoder Centroid	0.7	0.67
	F. IMLE-Encoder	**0.71**	0.67
	F. IMLE-Encoder Centroid	**0.71**	0.67
	ClusterGAN	**0.71**	**0.68**
Pendigits Odd Numbers	Simple IMLE	0.65	0.48
	IMLE 2-s N.N	**0.76**	**0.57**
	IMLE-Encoder	**0.76**	**0.57**
	IMLE-Encoder Centroid	**0.76**	**0.57**
	F. IMLE-Encoder	**0.76**	**0.57**
	F. IMLE-Encoder Centroid	**0.76**	**0.57**
	ClusterGAN	**0.76**	**0.57**
Pendigits Even Numbers	Simple IMLE	0.87	0.66
	IMLE 2-s N.N	0.87	0.77
	IMLE-Encoder	0.88	0.79
	IMLE-Encoder Centroid	0.87	0.78
	F. IMLE-Encoder	0.87	0.78
	F. IMLE-Encoder Centroid	0.88	0.79
	ClusterGAN	**0.96**	**0.89**

Synthetic Dataset

Table 6.12: Statistical analysis of the Accuracy on Synthetic Data.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	1.0	1.0	0.0	1.0	1.0
IMLE-Encoder Centroid	1.0	1.0	0.0.	1.0	1.0
IMLE-Encoder	1.0	1.0	0.0	1.0	1.0
F. IMLE-Encoder Centroid	1.0	1.0	0.0	1.0	1.0
F. IMLE-Encoder	1.0	1.0	0.0	1.0	1.0

Table 6.13: Statistical analysis of the NMI on Synthetic Data.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	1.0	1.0	0.0	1.0	1.0
IMLE-Encoder Centroid	1.0	1.0	0.0.	1.0	1.0
IMLE-Encoder	1.0	1.0	0.0	1.0	1.0
F. IMLE-Encoder Centroid	1.0	1.0	0.0	1.0	1.0
F. IMLE-Encoder	1.0	1.0	0.0	1.0	1.0

MNIST Dataset

Table 6.14: Statistical analysis of the Accuracy on MNIST dataset.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.5	0.5	0.03	0.44	0.59
IMLE-Encoder Centroid	0.51	0.52	0.03	0.4	0.57
IMLE-Encoder	0.51	0.52	0.03	0.4	0.55
F. IMLE-Encoder Centroid	0.71	0.71	0.03	0.66	0.78
F. IMLE-Encoder	0.71	0.71	0.03	0.66	0.79

Table 6.15: Statistical analysis of the NMI on MNIST dataset.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.44	0.44	0.03	0.38	0.51
IMLE-Encoder Centroid	0.43	0.45	0.04	0.31	0.49
IMLE-Encoder	0.44	0.44	0.04	0.32	0.50
F. IMLE-Encoder Centroid	0.44	0.44	0.03	0.39	0.50
F. IMLE-Encoder	0.44	0.43	0.04	0.33	0.52

MNIST Dataset, 4 Classes

Table 6.16: Statistical analysis of the Accuracy on MNIST dataset with 4 classes.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.84	0.87	0.06	0.64	0.89
IMLE-Encoder Centroid	0.81	0.85	0.09	0.59	0.91
IMLE-Encoder	0.81	0.81	0.1	0.59	0.92
F. IMLE-Encoder Centroid	0.81	0.86	0.09	0.59	0.89
F. IMLE-Encoder	0.8	0.86	0.1	0.56	0.91

Table 6.17: Statistical analysis of the NMI on MNIST dataset with 4 classes.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.64	0.67	0.07	0.42	0.70
IMLE-Encoder Centroid	0.61	0.1	0.38	0.31	0.76
IMLE-Encoder	0.61	0.63	0.1	0.38	0.77
F. IMLE-Encoder Centroid	0.61	0.65	0.08	0.39	0.69
F. IMLE-Encoder	0.61	0.67	0.1	0.33	0.73

Fashion-MNIST Dataset

Table 6.18: Statistical analysis of the Accuracy on Fashion-MNIST.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.56	0.55	0.03	0.5	0.63
IMLE-Encoder Centroid	0.57	0.58	0.03	0.5	0.64
IMLE-Encoder	0.57	0.56	0.03	0.5	0.64
F. IMLE-Encoder Centroid	0.56	0.56	0.03	0.5	0.63
F. IMLE-Encoder	0.55	0.55	0.04	0.45	0.63

Table 6.19: Statistical analysis of the NMI on Fashion-MNIST dataset.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.52	0.52	0.02	0.48	0.56
IMLE-Encoder Centroid	0.54	0.54	0.03	0.44	0.58
IMLE-Encoder	0.54	0.55	0.03	0.45	0.60
F. IMLE-Encoder Centroid	0.53	0.53	0.02	0.48	0.57
F. IMLE-Encoder	0.51	0.52	0.03	0.42	0.57

Fashion-MNIST Dataset, 5 Classes

Table 6.20: Statistical analysis of the Accuracy on Fashion-MNIST with 5 classes.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.66	0.66	0.02	0.62	0.7
IMLE-Encoder Centroid	0.68	0.67	0.02	0.62	0.73
IMLE-Encoder	0.7	0.68	0.03	0.63	0.79
F. IMLE-Encoder Centroid	0.66	0.67	0.02	0.61	0.71
F. IMLE-Encoder	0.67	0.67	0.03	0.6	0.73

Table 6.21: Statistical analysis of the NMI on Fashion-MNIST with 5 classes.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.48	0.5	0.05	0.39	0.56
IMLE-Encoder Centroid	0.51	0.50	0.04	0.39	0.59
IMLE-Encoder	0.53	0.52	0.04	0.42	0.60
F. IMLE-Encoder Centroid	0.49	0.49	0.03	0.42	0.57
F. IMLE-Encoder	0.49	0.49	0.03	0.39	0.56

Pendigits

Table 6.22: Statistical analysis of the Accuracy on Pendigits.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.7	0.71	0.03	0.64	0.76
IMLE-Encoder Centroid	0.7	0.7	0.03	0.61	0.76
IMLE-Encoder	0.71	0.7	0.03	0.61	0.76
F. IMLE-Encoder Centroid	0.71	0.71	0.03	0.66	0.78
F. IMLE-Encoder	0.71	0.71	0.03	0.66	0.79

Table 6.23: Statistical analysis of the NMI on Pendigits.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.66	0.66	0.03	0.6	0.71
IMLE-Encoder Centroid	0.67	0.66	0.03	0.6	0.72
IMLE-Encoder	0.67	0.67	0.03	0.6	0.71
F. IMLE-Encoder Centroid	0.67	0.67	0.02	0.63	0.71
F. IMLE-Encoder	0.67	0.68	0.02	0.63	0.72

Pendigits, Odd Numbers

Table 6.24: Statistical analysis of the Accuracy on Pendigits with odd numbers.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.76	0.76	0.009	0.75	0.78
IMLE-Encoder Centroid	0.76	0.76	0.007	0.75	0.78
IMLE-Encoder	0.76	0.76	0.009	0.75	0.79
F. IMLE-Encoder Centroid	0.76	0.76	0.01	0.75	0.79
F. IMLE-Encoder	0.76	0.76	0.01	0.74	0.79

Table 6.25: Statistical analysis of the NMI on Pendigits with odd numbers.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.57	0.56	0.02	0.55	0.62
IMLE-Encoder Centroid	0.57	0.57	0.01	0.55	0.6
IMLE-Encoder	0.57	0.57	0.02	0.55	0.64
F. IMLE-Encoder Centroid	0.57	0.56	0.02	0.55	0.60
F. IMLE-Encoder	0.57	0.56	0.02	0.54	0.62

Pendigits, Even Numbers

Table 6.26: Statistical analysis of the Accuracy on Pendigits with even numbers.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.87	0.88	0.05	0.64	0.91
IMLE-Encoder Centroid	0.88	0.89	0.03	0.8	0.92
IMLE-Encoder	0.88	0.89	0.03	0.8	0.92
F. IMLE-Encoder Centroid	0.87	0.89	0.04	0.71	0.92
F. IMLE-Encoder	0.87	0.89	0.04	0.72	0.91

Table 6.27: Statistical analysis of the NMI on Pendigits with even numbers.

Algorithm	Mean	Median	St. Dev.	Min	Max
IMLE 2-s N.N	0.77	0.78	0.04	0.6	0.81
IMLE-Encoder Centroid	0.79	0.79	0.03	0.7	0.83
IMLE-Encoder	0.79	0.79	0.03	0.7	0.83
F. IMLE-Encoder Centroid	0.78	0.79	0.04	0.68	0.82
F. IMLE-Encoder	0.78	0.79	0.04	0.68	0.83

6.4 Image Generation

We utilized the Generator in order to cluster the data in the latent space. We can also use the trained Generator to generate synthetic samples that look similar to the training data. We except that if the clustering was successful, the trained Generator would assign each data cluster to exactly one Gaussian component. In figure 6.1 we present the synthetic samples created by the Generator, which is trained at the MNIST with four clusters. The samples of the first image (a) are created by the first Gaussian component, of the second image (b) from the second component, etc. In figure 6.1 we also present the synthetic samples created by the Generator, which is trained at the fashion-MNIST with five clusters, and the samples are created the same way we already explained. Thus, we can conclude that successful data clustering can also result in successful synthetic samples clustering at the latent space of the Generator as well.

As we already presented in figure 5.1, the latent space in a traditional IMLE with Gaussian latent distribution enforces that different classes are continuously scattered in the latent space. In order to demonstrate interpolation at the latent space, we sampled the latent vector z_c by a one-hot distribution, and we used a linear combination of it to interpolate across the clusters. More specifically, we fixed the z_n in two latent vectors with different z_c components, we defined $z_c^{(1)}$ and $z_c^{(2)}$ and then we interpolated with the one-hot encoded part to give rise to new latent vectors $z = (z_n, \mu z_c^{(1)} + (1{-}\mu) z_c^{(2)})$, $\mu \in [0, 1]$. While these vectors have never been sampled during the training process, we observed a smooth inter-class interpolation. As Figure 6.3 illustrates, we observed a nice transition from one digit to another at MNIST dataset as well as across different classes in Fashion-MNIST. This demonstrates that our IMLE based approach learns a smooth manifold even on the untrained directions of the discrete-continuous distribution.

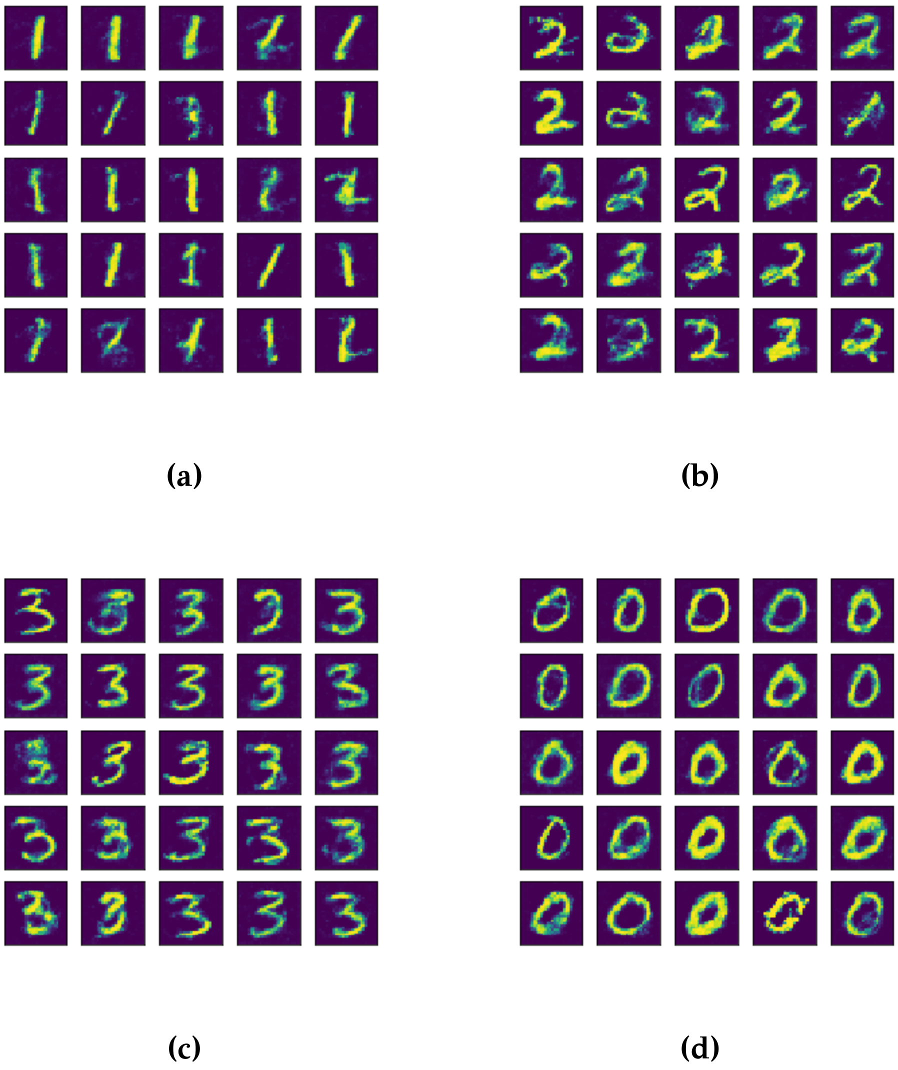
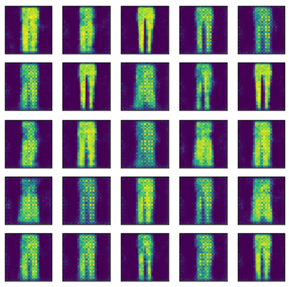
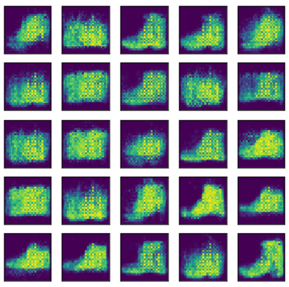
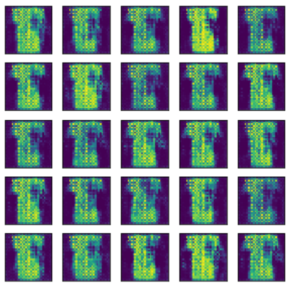
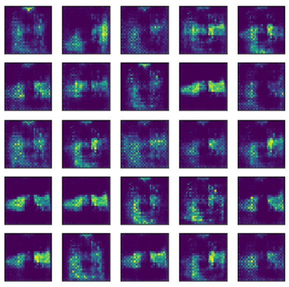


Figure 6.1: Digits generated from distinct modes (MNIST with four classes).


(a)


(b)


(c)


(d)

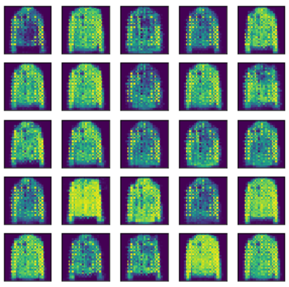

(e)

Figure 6.2: Digits generated from distinct modes (Fashion-MNIST with five classes).

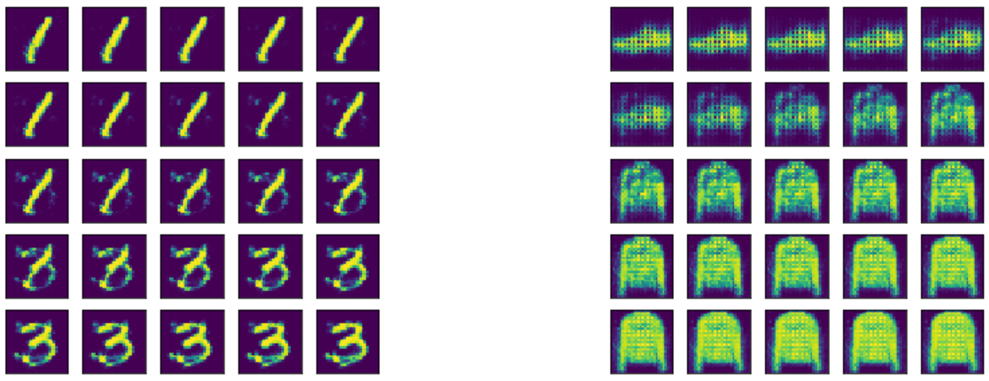


Figure 6.3: Latent Space Interpolation: MNIST (left) and Fashion-MNIST (right).

6.5 Dimensionality Reduction

After training, the Encoder network can cluster the data and reduce their dimensionality by mapping it to the latent space. To verify the quality of the dimensionality reduction the Encoder can achieve, we will apply TSNE [4] to the Encoder's output to lower its manifold to the two-dimensional space. Figure 6.4 presents the two-dimensional projection of the Fashion-MNIST dataset with five classes, and figure 6.5 presents the two-dimensional projection of the MNIST dataset with four classes. We can note that the different clusters are projected separated areas of the two-dimensional space, which declares the Encoder's dimensionality reduction quality. Finally, we can see in figures 6.6, 6.7 that the Encoder was able to provide a dimensionality reduction of such good quality for the Synthetic data that every data cluster belongs to a different area in the two-dimensional space without any cluster overlap.

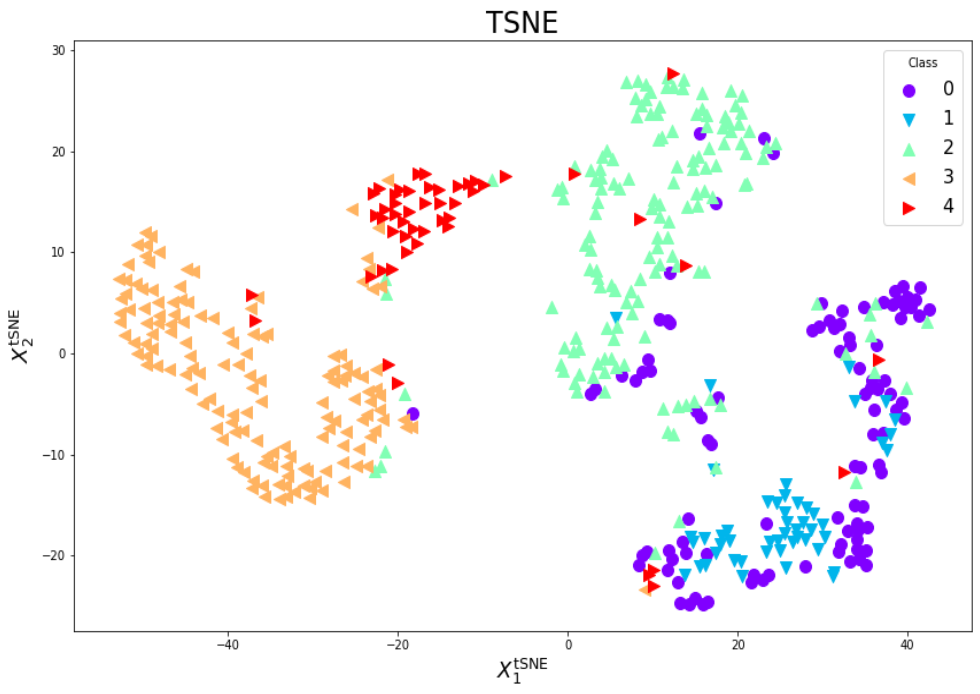


Figure 6.4: Lowering Encoder's output manifold, using TSNE [4] at Fashion-MNIST dataset with five classes.

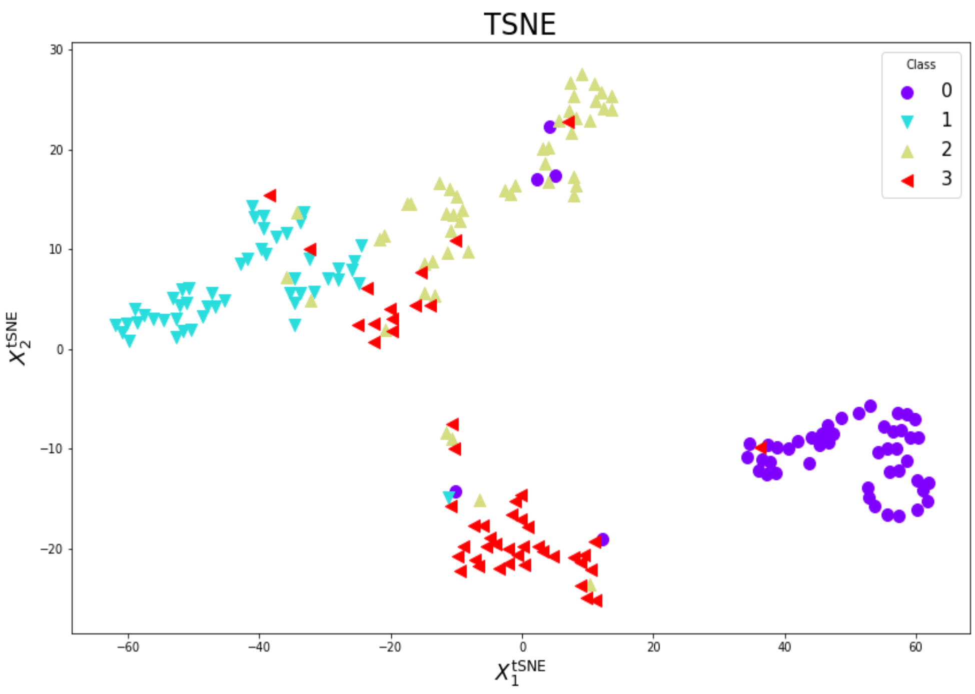


Figure 6.5: Lowering Encoder's output manifold, using TSNE [4] at MNIST dataset with four classes.

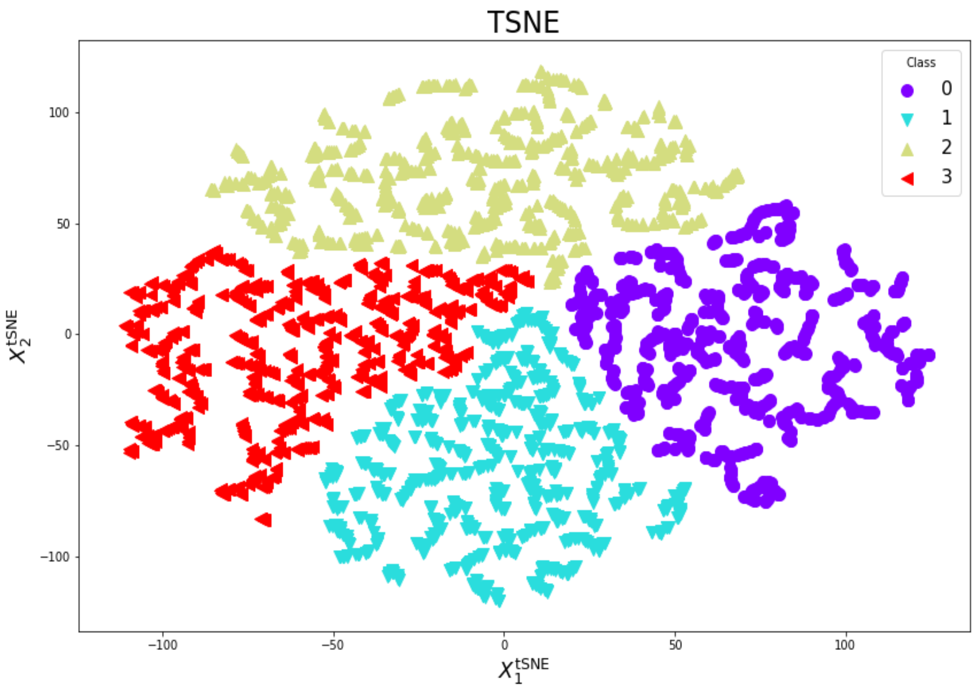


Figure 6.6: Lowering Encoder's output manifold, using TSNE [4] (perplexity=10) at Synthetic dataset with four classes.

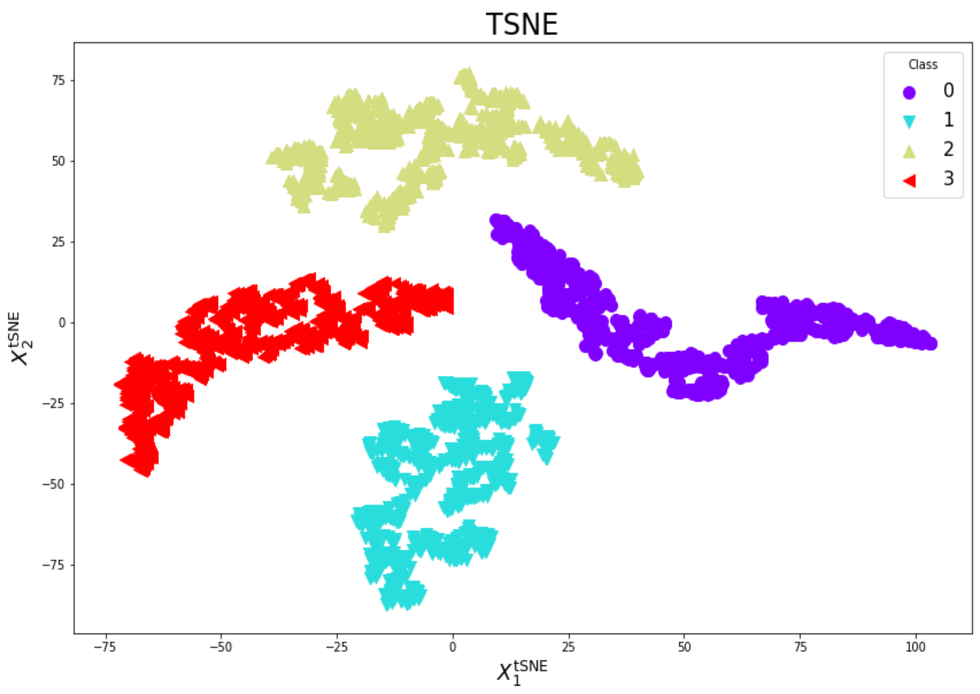
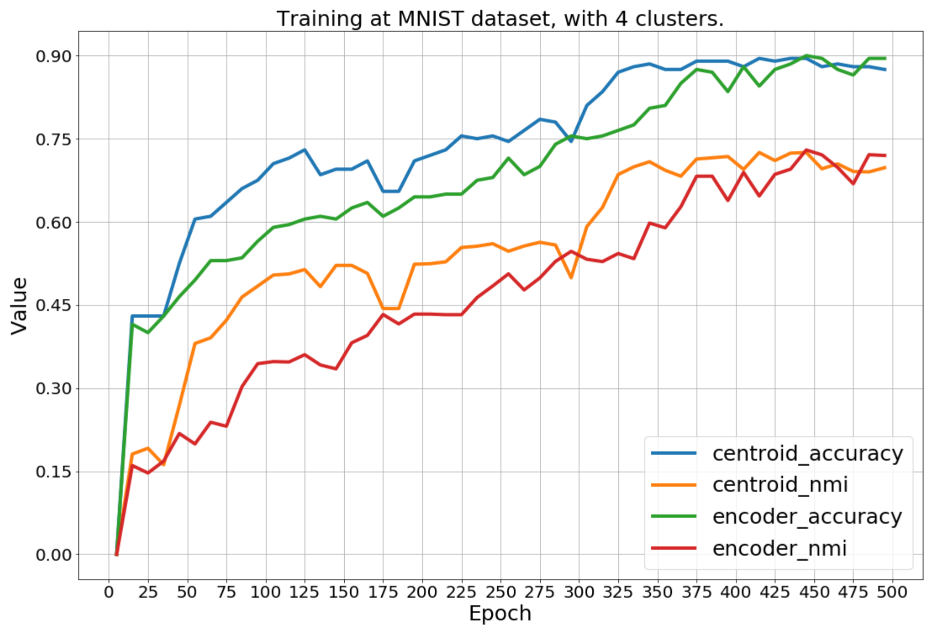


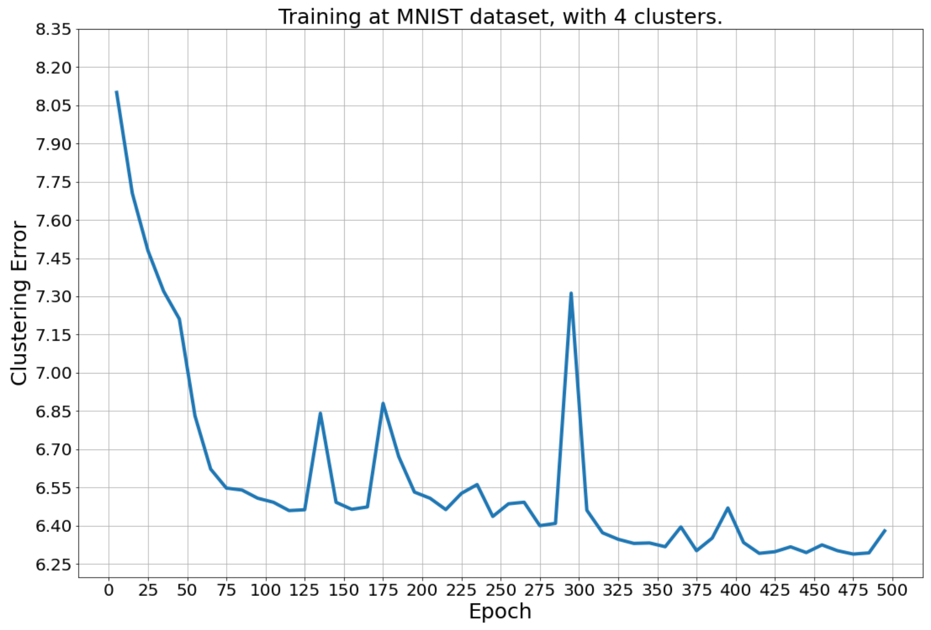
Figure 6.7: Lowering Encoder's output manifold, using TSNE [4] (perplexity=30) at Synthetic dataset with four classes.

6.6 Training Procedure

As figure 6.8 (a) illustrates, during the IMLE based model training, we can see that the Accuracy and the NMI metrics tend to improve but not monotonically since fluctuations exist. The same is true for the clustering error (figure 6.8 (b)). It worth noting that the best clustering is not necessarily the one with the minimum clustering error, especially when the training data are complex like images.



(a)



(b)

Figure 6.8: (a) Accuracy and normalized mutual information metrics based on centroid and encoder (b) Clustering error per epoch.

Chapter 7

Epilogue

7.1 Conclusion
7.2 Suggestions for Future Work

7.1 Conclusion

In this thesis, we studied clustering using deep neural networks (called deep clustering) and proposed a new deep clustering method, the Deep Clustering based on IMLE. The method is based on IMLE approach, which is a technique of maximizing the likelihood of the model without including any likelihood term in the objective function. The experimental results show that the Deep Clustering based on IMLE can generate synthetic samples using a generators network with a minimum number of training data. As we observed using image datasets, the generated samples are of good quality since they resemble the training data. Additionally, the created manifold provides a very smooth inter-class interpolation. Furthermore, using the encoder network, we can perform data clustering at the latent space and provide an exceptional method of dimensionality reduction.

7.2 Suggestions for Future Work

As seen from the experimental results, we can achieve reasonable performance with our current implementation. However, there is a lot of reasearch work to be done:

- Conduct experiments with the complete dataset to evaluate the IMLE based method's full potential. Additional experiments can take place with different image datasets.

- It is interesting to experiment with a pre-trained Generator and Encoder to determine if the results will improve. Pre-training is a common tactic in Deep Learning and tends to have better results than random initialization of the parameters. For example, many Deep Clustering methods use pre-training DNNs in order to get better results [39, 42, 43, 44, 62].

- It is worth examining more hyperparameters setups in order to fine-tune the existing model.

- Performing experiments with more complex training schedules. For example, at the beginning of the model's training, the generative loss could have greater value from the clustering loss. Then, the generative loss could start decaying as the training continues, and the clustering loss to stat rising.

- It would be interesting to execute experiments with more sophisticated deep features and take advantage of the complex deep features the Encoder DNN can compute.

- It is worth mentioning that the whole procedure can occur in the Encoder's deep feature space. It would be interesting to experiment with the transformed training data from the data space X_r to the Encoder's deep feature space X_r^f. After that we could find the centroids $C_{x_s}^f$ which are constructed by the deep features of the samples X_s^f and then finding the nearest sample $x_{s_j}^f$ to each training data $x_{r_i}^f$.

- We executed experiments with more Gaussian components than the number of clusters in the data, and they had promising results. It would be interesting to train the model with more Gaussian components. Then, as the training proceeds, we could decrease the Gaussian components until their number equals the desirable number of clusters.

- It is worth studying for a convergence criterion to determine when the to stop learning.

Bibliography

[1] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, “Clustergan: Latent space clustering in generative adversarial networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4610–4617.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[3] K. Li and J. Malik, “Implicit maximum likelihood estimation,” *arXiv preprint arXiv:1809.09087*, 2018.

[4] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.” *Journal of machine learning research*, vol. 9, no. 11, 2008.

[5] I. Kononenko, “Machine learning for medical diagnosis: history, state of the art and perspective,” *Artificial Intelligence in medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[6] E. B. Hunt, J. Marin, and P. J. Stone, “Experiments in induction.” 1966.

[7] N. J. Nilsson, “Learning machines.” 1965.

[8] F. Rosenblatt, “Principles of neurodynamics. perceptrons and the theory of brain mechanisms,” Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

[9] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, “Machine learning, neural and statistical classification,” 1994.

[10] F. Rosenblatt, “Perceptron simulation experiments,” *Proceedings of the IRE*, vol. 48, no. 3, pp. 301–309, 1960.

[11] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[13] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14 413–14 423, 2020.

[14] A. B. Novikoff, "On convergence proofs for perceptrons," STANFORD RESEARCH INST MENLO PARK CA, Tech. Rep., 1963.

[15] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[17] S. Mohamed and B. Lakshminarayanan, "Learning in implicit generative models," *arXiv preprint arXiv:1610.03483*, 2016.

[18] C. M. Bishop, "Pattern recognition," *Machine learning*, vol. 128, no. 9, 2006.

[19] R. Xu and D. Wunsch, *Clustering*. John Wiley & Sons, 2008, vol. 10.

[20] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[21] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.

[22] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, "A survey of kernel and spectral methods for clustering," *Pattern recognition*, vol. 41, no. 1, pp. 176–190, 2008.

[23] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[24] S. Theodoridis and K. Koutroumbas, "Chapter 11 - clustering: Basic concepts," in *Pattern Recognition (Fourth Edition)*, fourth edition ed., S. Theodoridis and K. Koutroumbas, Eds. Boston: Academic Press, 2009, pp. 595–625. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978159749272050013X

[25] A. Cornuéjols, C. Wemmert, P. Gançarski, and Y. Bennani, "Collaborative clustering: Why, when, what and how," *Information Fusion*, vol. 39, pp. 81–95, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253517300027

[26] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[27] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, "Clustering large graphs via the singular value decomposition," *Machine learning*, vol. 56, no. 1, pp. 9–33, 2004.

[28] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Icml*, 2011.

[29] G. Nutakki, B. Abdollahi, W. Sun, and O. Nasraoui, *An Introduction to Deep Clustering*, 01 2019, pp. 73–89.

[30] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, "A survey of clustering with deep learning: From the perspective of network architecture," *IEEE Access*, vol. 6, pp. 39 501–39 514, 2018.

[31] E. Aljalbout, V. Golkov, Y. Siddiqui, M. Strobel, and D. Cremers, "Clustering with deep learning: Taxonomy and new methods," *arXiv preprint arXiv:1801.07648*, 2018.

[32] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2010.

[33] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[34] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 609–616.

[35] G. E. Hinton, "A practical guide to training restricted boltzmann machines," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 599–619.

[36] ——, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009.

[37] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[38] S. Saito and R. T. Tan, "Neural clustering: Concatenating layers for better projections," 2017.

[39] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *international conference on machine learning*. PMLR, 2017, pp. 3861–3870.

[40] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*. PMLR, 2016, pp. 478–487.

[41] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 407–416.

[42] C.-C. Hsu and C.-W. Lin, "Cnn-based joint clustering and representation learning with feature drift compensation for large-scale image data," *IEEE Transactions on Multimedia*, vol. 20, no. 2, pp. 421–429, 2017.

[43] G. Chen, "Deep learning with nonparametric clustering," *arXiv preprint arXiv:1501.03084*, 2015.

[44] P. Huang, Y. Huang, W. Wang, and L. Wang, "Deep embedding network for clustering," in *2014 22nd International conference on pattern recognition*. IEEE, 2014, pp. 1532–1537.

[45] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5147–5156.

[46] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[47] F. Li, H. Qiao, and B. Zhang, "Discriminatively boosted image clustering with fully convolutional auto-encoders," *Pattern Recognition*, vol. 83, pp. 161–173, 2018.

[48] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics (NRL)*, vol. 52, no. 1, pp. 7–21, 2005.

[49] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada, "Normalized mutual information feature selection," *IEEE Transactions on neural networks*, vol. 20, no. 2, pp. 189–201, 2009.

[50] A. Jalal, A. Ilyas, C. Daskalakis, and A. G. Dimakis, "The robust manifold defense: Adversarial training using generative models," *arXiv preprint arXiv:1712.09196*, 2017.

[51] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, "Compressed sensing using generative models," in *International Conference on Machine Learning*. PMLR, 2017, pp. 537–546.

[52] S. Santurkar, D. Budden, and N. Shavit, "Generative compression," in *2018 Picture Coding Symposium (PCS)*. IEEE, 2018, pp. 258–262.

[53] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[54] K. Li and J. Malik, "Fast k-nearest neighbour search via dynamic continuous indexing," in *International Conference on Machine Learning*, 2016, pp. 671–679.

[55] ——, "Fast k-nearest neighbour search via prioritized dci," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2081–2090.

[56] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, pp. 2234–2242, 2016.

[57] A. Dosovitskiy and T. Brox, "Generating images with perceptual similarity metrics based on deep networks," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/371bce7dc83817b7893bcdeed13799b5-Paper.pdf

[58] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[59] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford InfoLab, Technical Report 2006-13, June 2006. [Online]. Available: http://ilpubs.stanford.edu:8090/778/

[60] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[61] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[62] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 132–149.

[63] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[65] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[67] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[68] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[69] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

Appendix A

Software requirements

The thesis code was written in Python [63] programming language. The following are the necessary python libraries used to implement the program.

- PyTorch [64] is an open-source machine learning framework. PyTorch is an exceptionally useful and flexible libraries for the constructing and training of artificial neural network architecture.
- Numpy [65] library is fundamental package for scientific computing with Python. Numpy is essential for scientific computations and matrices' creation and manipulation.
- Scikit-learn [66] is a simple and efficient library that contains tools for machine learning, data analysis, and scientific computations.
- SciPy [67] is an open-source library used for mathematics, science, and engineering.
- Pandas [68] is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool.
- Matplotlib [69] is a comprehensive library for creating static, animated, and interactive visualizations.

Short Biography

Georgios Vardakas was born in Ioannina, Greece, in 1995. In 2014 he enrolled in the undergraduate program of Computer Science and Engineering of the University of Ioannina and earned his Diploma in 2020. His Diploma thesis was entitled ”Deep Learning based on Implicit Maximization of Likelihood.” In 2020 he enrolled in the post-graduate program of the same Department. His research interests focus on Machine Learning and, more specifically, on Deep Learning methods.