A Silhouette-Based Deep Clustering Method

I. Papakostas

Master Thesis

— ♦ —

Ioannina, February 2025

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

A Silhouette Based Deep Clustering Method

A Thesis

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

by

Ioannis Papakostas

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 2025

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 Aristidis Likas, for for the valuable guidance, patience and trust he has shown me throughout our collaboration. He was more than eager to share his knowledge on the subject while making helpful suggestions throughout this thesis. His guidance, support, and expertise were only some of the ways in which he shaped my first steps in research.

I would also like to thank the PhD candidate Georgios Vardakas for his valuable help in the implementation of this thesis and the experiments that should be done. He was there for me when I needed help and he had some very good ideas about my thesis topic.

I would also like to thank my family for their unconditional support and love over the years. My mother, Maria, who has always supported me, my brother Vasilis, and my father, Stathis, who taught me to try as hard as possible to achieve my goals. Moreover, I wish to thank my girlfriend Angeliki-Maria and my close friends Panagiotis and Aris for their support and understanding during my studies.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Glossary

AE Autoencoder

ANN Artificial Neural Network

ARI Adjusted Rand Index

BD EMNIST Balanced Digits

BL EMNIST Balanced Letters

CNN Convolutional Neural Network

DBN Deep Belief Network

DCN Deep Clustering Network

DCSS Deep Clustering using Soft Silhouette

DEC Deep Embedded Clustering

DL Deep Learning

DNN Deep Neural Network

DR Dimensionality Reduction

EMNIST EMNIST MNIST

GAN Generative Adversarial Network

HAR Human Activity Recognition with Smartphones

IDEC Improved Deep Embedded Clustering

KL Kullback-Leibler

LapEig	Laplacian Eigenmap
LLE	Local Linear Embeddings
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NMF	Non-negative Matrix Factorization
NMI	Normalized Mutual Information
PCA	Principal Component Analysis
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machines
ReLU	Rectified Linear Unit
SAE	Stacked Autoencoder
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition
VAE	Variational Autoencoder
WVF-v1	Waveform-v1

Abstract

Ioannis Papakostas, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2025.
A Silhouette Based Deep Clustering Method.
Advisor: Aristidis Likas, Professor.

In the context of big data, unsupervised learning has emerged as a crucial field of study, offering methodologies for the discovery of insights from datasets lacking labels. Deep clustering has established itself as a prominent approach within the domain of unsupervised learning, capitalizing on the nonlinear mapping capabilities of neural networks to enhance clustering performance. A significant proportion of existing literature on deep clustering is dedicated to the minimization of within-cluster variability in an embedded space, while ensuring that the learned representation remains faithful to the original high-dimensional dataset.

This thesis elaborates on a novel approach, designated as soft silhouette, which presents a probabilistic formulation of the silhouette coefficient. As with the traditional silhouette coefficient, the soft silhouette encourages the formation of compact and well-separated clusters. When integrated into a deep clustering framework, the soft silhouette approach guides the learning process towards the creation of compact and distinct clusters.

The deep clustering method has been subjected to rigorous testing and comparison with several established deep clustering methods using a variety of benchmark datasets. The results demonstrate that the silhouette based deep clustering approach yields highly satisfactory clustering outcomes.

Keywords: machine learning, clustering, deep clustering, soft Silhouette score, artificial neural network, loss function, autoencoder, convolutional-autoencoder, representation learning

Εκτεταμενη Περιληψη

Ιωάννης Παπακώστας, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2025.
Μια Μέθοδος Βαθιάς Ομαδοποίησης Βασισμένη στο Silhouette.
Επιβλέπων: Αριστείδης Λύκας, Καθηγητής.

Στην εποχή των μεγάλων δεδομένων, η μάθηση χωρίς επίβλεψη γίνεται όλο και πιο σημαντική, προσφέροντας έναν τρόπο εύρεσης πολύτιμων πληροφοριών σε σύνολα δεδομένων που δεν έχουν ετικέτες, δηλαδή δεν συνοδεύονται από προκαθορισμένες κατηγορίες. Η βαθιά ομαδοποίηση έχει αναδειχθεί ως μια σημαντική προσέγγιση στο πλαίσιο της μάθησης χωρίς επίβλεψη, αξιοποιώντας τις δυνατότητες της μη γραμμικής απεικόνισης που προσφέρουν τα νευρωνικά δίκτυα για τη βελτίωση της απόδοσης της ομαδοποίησης. Μεγάλο μέρος της υπάρχουσας βιβλιογραφίας για τη βαθιά ομαδοποίηση εστιάζει στην ελαχιστοποίηση της μεταβλητότητας εντός της ομάδας - συστάδας σε έναν λανθάνον χώρο, σημαντικά μικρότερης διάστασης από τον αρχικό, διασφαλίζοντας παράλληλα ότι η εκμάθηση της αναπαράστασης παραμένει πιστή στο αρχικό σύνολο δεδομένων.

Η εργασία αυτή μελετά μια καινοτόμα προσέγγιση που βασίζεται στο κριτήριο soft silhouette και αποτελεί μια πιθανολογική διατύπωση της μετρικής silhouette. Παρόμοια με τον παραδοσιακό συντελεστή silhouette, ο soft silhouette ενθαρρύνει το σχηματισμό συμπαγών και καλά διαχωρίσιμων συστάδων. Όταν ενσωματώνεται σε ένα πλαίσιο βαθιάς ομαδοποίησης, ο soft silhouette καθοδηγεί τη διαδικασία της μάθησης προς τη δημιουργία ομάδων που είναι πυκνές και διακριτές μεταξύ τους. Επιπλέον, παρουσιάζεται μια αρχιτεκτονική βαθιάς μάθησης που βασίζεται σε autoencoder και βελτιστοποιεί αποτελεσματικά τη λειτουργία της αντικειμενικής συνάρτησης soft silhouette.

Η εξεταζόμενη μέθοδος βαθιάς ομαδοποίησης υποβλήθηκε σε εκτεταμένες δοκιμές και σύγκριση με διάφορες καθιερωμένες μεθόδους βαθιάς ομαδοποίησης μέσα από την χρήση πολλών συνόλων δεδομένων αναφοράς. Τα αποτελέσματα δείχνουν ότι η μέθοδος προσφέρει ιδιαίτερα ικανοποιητικά αποτελέσματα ομαδοποίησης.

Λέξεις-κλειδιά: μηχανική μάθηση, ομαδοποίηση, βαθιά ομαδοποίηση, soft Silhouette σκορ, τεχνητό νευρωνικό δίκτυο, συνάρτηση απώλειας, autoencoder, convolutional-autoencoder, εκμάθηση αναπαράστασης

Chapter 1

Introduction

1.1 Machine Learning

Machine learning is a subject that researches how to use computers to simulate human learning activities and to study self-improvement methods of computers to acquire new knowledge and skills, recognize existing knowledge, and continuously improve performance and achievement.

Compared with human learning, machine learning learns faster, the accumulation of knowledge is more facilitated and the results of learning spread easier. So, any progress of humanity in the field of machine learning will enhance the capability of computers, thus it will have an impact on human society [1].

It is the field of artificial intelligence that deals with the study of algorithms that analyze data, recognize patterns in data, and improve their performance by interacting with data. Thus, they make decisions without human intervention. Algorithms used in machine learning should have good general performance. They must perform well even for examples they have not trained for but follow the same distribution as the training examples.

There are several categories of machine learning algorithms, but the most important are:

- **Supervised learning**: Such algorithms are trained with data that include for each data point an input vector x as well as the desired output y. Using these examples, they learn to predict the output y for any input x, usually by evaluating the probability $P(y|x)$. Common supervised learning problems are classification and function approximation problems.
- **Unsupervised learning**: These algorithms are provided with training data that contain for each data point only one input vector x. They discover patterns and correlations that exist in the data. Usual unsupervised learning problems are clustering, data dimensionality reduction (DR) etc. The main focus of this master thesis focuses on this kind of machine learning problems.
- **Reinforcement learning**: Here the algorithm is trained by making decisions to maximize some reward. The algorithm chooses the sequence that maximizes the reward and thus composes its policy.

1.2 Clustering

Clustering is the distinction of data into groups so that data belonging to the same group show great similarity to each other, while data belonging to different groups show great dissimilarity. Depending on the information of data being clustered, various similarity measures could be used.

1.2.1 Hierarchical Clustering Algorithms

The hierarchical clustering algorithms produce a grouping structure in the form of a tree, each node of which is a group. As a result, we have different numbers of groups depending on the level of the tree structure. The root of the tree is a group to which all data belong. Each node has two children which are the groups that result if we split the parent group into two. The leaves of the tree are the non-clustered data. There are two basic hierarchical algorithms:

- **Divisive clustering**: Starts with all examples in one group and starts dividing the group with the largest internal distance into two groups until we have the desired number of groups.
- **Cumulative clustering**: Starts with each instance as a different cluster and joins the two clusters with the shortest distance until we have the desired number of clusters.

Hierarchical algorithms typically use the following three distance metrics between clusters:

- **Single linkage**: As distance between two groups is considered the minimum of the intervals between the members of one group and the members of the other. More specifically, it is the distance between the closest members of the two groups.
- **Complete linkage**: As distance between two groups is considered the maximum of the intervals between the members of one group and the members of the other. More precisely, it is the distance between the most distant members of the two groups. An algorithm that uses complete linkage tends to produce more compact clusters than using the above one. However, single linkage could achieve more complex clustering [2].
- **Average linkage**: As distance between two groups is considered the average value of the intervals between the members of one group and the members of the other.

1.2.2 k-means Algorithm

Given a collection of data samples $\{x_i\}_{i=1}^N$, where $x_i \in \mathbb{R}^M$, the objective of clustering is to partition the N data samples into K categories. k-means, proposed by Lloyd in 1982 [3], stands as one of the most widely adopted algorithms for this task. k-means seeks to minimize the following cost function:

$$\min_{M \in \mathbb{R}^{M \times K}, s_i \in \mathbb{R}^K} \sum_{i=1}^{N} \|x_i - Ms_i\|_2^2 \tag{1.1}$$

subject to $s_{j,i} \in \{0, 1\}$ and $1^\top s_i = 1$ for $\forall i, j$, where s_i is the assignment vector of data point i which has only one non-zero element, $s_{j,i}$ denotes the jth element of s_i, and the kth column of M, i.e., m_k, denotes the centroid of the kth cluster.

The algorithm is trained iteratively according to the following steps:

1. **Initialization of k centers.** k-means is highly dependent on the initialization of the centers in whether it will converge to good clustering. Various initialization techniques have been developed for solving this problem. Some common and typical methods of initializing centers are [4, 5]:

   - Select k random samples from the data as centers. Thus, it is more likely to select examples that are in areas with high density. Such examples are a good initial choice for group centers. But the selection of remote examples could not be ruled out. As a result, a good technique is to repeat the initialization several times.
   - Assigning the data to k groups randomly and using the centers as the initial means of the k groups.
   - Picking the first center randomly from the data examples and then choosing each subsequent center as the instance that is the furthest distance from the selected center nearest to it.
   - Picking the first center randomly from the data examples and then choosing each subsequent center as the example x_i with probability:

     $$\frac{md(x_i)^2}{\sum_{j=1}^{N} md(x_j)^2}, \tag{1.2}$$

     where $md(x)$ is the distance of sample x from the nearest selected center. This method is called k-means++.

2. For each data instance, we calculate its distance from k centers and assign it to the group from whose center has the shortest distance.

3. We update the k centers as the mean value of the examples assigned to their group according to the relation:

   $$c_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i, \tag{1.3}$$

where c_j is the new center of the jth group, and S_j is the set of examples assigned to the jth group in this step.

4. Repeat steps 2 and 3 until convergence, meaning that the updated centroids do not change positions.

k-means performs well, when data samples are evenly distributed around their centroids in the feature space. However, high-dimensional data typically do not exhibit characteristics that are conducive to k-means clustering [6]. Finally, k-means has a very high dependence on its initialization and on examples that are a long distance from the rest (outliers). These instances will be assigned to a group and will affect its center [7].

1.2.3 Fuzzy Clustering

In fuzzy clustering, each example can belong to several groups and is labeled by its degree of participation in each cluster. The degree of this participation takes values between 0 and 1. It expresses how much the given example can be considered as a group member. So, for example, an instance on the border between two groups has equal participation scores close to 0.5 for both groups, but an example in the center of a group has a very high participation score in that group and lower in the others. The advantage of these algorithms is that they quantify and incorporate into the output the doubt they have as to which group an example belongs.

Fuzzy C-Means

The algorithm assigns a participation score to each example for each of the k groups. The training for Fuzzy C-means is achieved according to the following steps [8]:

1. Initialize the matrix U whose elements c_{ij} are the degree of participation of example i in the group j. This initialization is usually done with random values in the interval [0, 1]. Also, there is here, as in k-means, a strong dependence on the initialization, so the algorithm is often executed several times with different initialization.

2. Calculation of the centers of k groups. The center of group j will be $c_j =$

$$\frac{\sum_{i=1}^{N} u_{ij}^m x_i}{\sum_{i=1}^{N} u_{ij}^m}, \tag{1.4}$$

   where N is the number of examples and $m \geq 1$ is a parameter that determines how fuzzy the clustering will be. Larger values of the parameter cause more fuzzy clustering, i.e. smaller values in the participation scores. For $m = 1$ the participation scores only take values 0 or 1 and we no longer have fuzzy clustering.

3. Renewal of participation scores u_{ij} according to the relation:

$$u_{ij} = \frac{1}{\sum_{n=1}^{k} \left(\frac{||x_i - c_j||}{||x_i - c_n||}\right)^{\frac{2}{m-1}}}. \tag{1.5}$$

4. As long as the termination criterion is not met, repeat steps 2 and 3. The most common termination criterion is to hold in two consecutive iterations n and $n+1$: $||U_{n+1} - U_n|| < \varepsilon$, where ε is a small real number, i.e. participation scores do not change significantly.

And the Fuzzy C-means algorithm is very dependent on the initialization and on examples that are far from the rest (outliers) [7].

EM (Expectation-Maximization)

Mixed distribution models assume that our data come from a mixture of different distributions. The probability that an example x_i could produce a particular mixture of k distributions, is given by the relation $p(x_i|\vartheta) = \sum_{j=1}^{k} \pi_j p_j(x_i|\vartheta)$, where π_j is the prior probability that the distribution j is responsible for producing the example x_i and $p_j(x_i|\vartheta)$ is the probability that x_i was generated by this distribution with parameters ϑ. We may have a mixture of distributions of different types or a mixture of distributions of the same type with different parameters. The most common case is the mixture of k Gaussian distributions N with different means μ and covariance matrices Σ. In this case, the above relation becomes $p(x_i|\vartheta) = \sum_{j=1}^{k} \pi_j N_j(x_i|\mu_j, \Sigma_j)$.

If we consider that each distribution of the model represents a different group, then from the above probabilities we can calculate the probability r_{ij} that the example

x_i belongs to the group j according to the relation $r_{ij} = \frac{\pi_j p_j(x_i|\vartheta)}{p(x_i|\vartheta)}$. Therefore we can use these probabilities to group our data by calculating the probability that each instance belongs to a group [9].

Training of the model is to find the parameters ϑ of the distributions that maximize $l(\vartheta) = \sum\limits_{i=1}^{N} \log p(x_i|\vartheta)$. The most common algorithm for this is **EM (Expectation-Maximization)** which has the following steps:

1. Initialize the parameters ϑ of the distributions to random values.
2. Step **E (Expectation)**: Calculation of the posterior probabilities r_{ij} that the example x_i belongs to the group j $\forall i, j$ according to the existing values of the parameters ϑ.
3. Step **M (Maximization)**: Calculation of the new $\pi_j = \frac{1}{N} \sum\limits_i r_{ij}$ and use them to update the values of the parameters ϑ in order to maximize $l(\vartheta)$.
4. Repeat steps 2 and 3 until the parameter values converge.

In the case where we have a mixture of k Gaussian distributions from the maximization of $l(\vartheta)$, the following relations for the parameters occur $\mu_j = \frac{\sum\limits_i r_{ij} x_i}{\sum\limits_i r_{ij}}$ and $\Sigma_j = \frac{\sum\limits_i r_{ij}(x_i-\mu_j)(x_i-\mu_j)^T}{\sum\limits_i r_{ij}}$. The **EM** algorithm has a strong dependence on the initialization of the ϑ parameters and the form of the covariance matrices [7].

1.3 Distance Metrics

A fundamental issue in the clustering problem is the selection of the appropriate distance metric to use in all algorithms. The distance metric is the way to quantify the similarity between two examples. Depending on the data format, algorithms are more efficient using a different distance metric.

Each distance metric must have the properties:

- Be **positive definite**, i.e. for any pair x_i, x_j, $d(x_i, x_j) \geq 0$.
- Be **symmetrical**, i.e. for any pair x_i, x_j, $d(x_i, x_j) = d(x_j, x_i)$.
- Be **identical**, i.e. $d(x_i, x_j) = 0 \Leftrightarrow x_i = x_j$

- Have the **triangular property**, i.e. for any x_i, x_j, x_k, $d(x_i, x_j) \leq d(x_i, x_k) + d(x_k, x_j)$.

However, metrics often used, don't meet all four criteria and don't have the triangular property.

Some of the most common distance metrics between numerical vectors are:

- **Euclidean** or **l2** distance: For two vectors x_i, x_j of dimension m, Euclidean distance is defined as $d(x_i, x_j) = \sqrt{\sum\limits_{k=1}^{m} (x_i^k - x_j^k)^2}$. It is the most classic distance metric and satisfies all of the above properties. In the two dimensions, it is the length of the straight segment that joins the points i, j. In more dimensions, it is the generalization of this metric.

- **Manhattan** or **l1** distance: For two vectors x_i, x_j of dimension m, Manhattan distance is defined as $d(x_i, x_j) = \sum\limits_{k=1}^{m} |x_i^k - x_j^k|$. It satisfies all of the above properties and expresses the distance from one vector to another by crossing along each dimension separately.

- **Chebyshev** or **l**$_\infty$ distance: For two vectors x_i, x_j of dimension m, Chebyshev distance is defined as $d(x_i, x_j) = \max\limits_k |x_i^k - x_j^k|$. It satisfies all of the above properties.

- **Minkowski** distance: For two vectors x_i, x_j of dimension m, Minkowski distance is defined as $d(x_i, x_j) = \sqrt[p]{\sum\limits_{k=1}^{m} |x_i^k - x_j^k|^p}$, where p is the order of the distance. Obviously, Euclidean distance is a special case of Minkowski distance for $p = 2$, Manhattan distance for $p = 1$, and Chebyshev distance for $p = \infty$. For $p \geq 1$, Minkowski distance satisfies all four properties. For $p < 1$, it does not satisfy the triangular property. By changing the value of p, we change the geometric locus of points that are equidistant from a point.

- **Cosine** distance: For two vectors x_i, x_j, cosine distance is defined as $d(x_i, x_j) = 1 - \frac{x_i x_j}{||x_i|| ||x_j||}$ and shows us how much the two vectors differ in direction. It takes values from 0, for vectors in the same direction, to 2, for vectors in the opposite direction.

- **Mahalanobis** distance: For two vectors x_i, x_j of a common distribution, Mahalanobis distance is defined as $d(x_i, x_j) = \sqrt{(x_i - x_j)^\intercal \Sigma^{-1} (x_i - x_j)}$, where Σ is the covariance matrix of the data. Depending on the form of the covariance matrix, we can find the distance of the points by giving a different weight to each dimension.

- **Pearson** correlation: Using the Pearson correlation of two vectors x_i, x_j, we can calculate a distance metric as $d(x_i, x_j) = 1 - |\frac{(x_i - \bar{x_i})(x_j - \bar{x_j})}{||(x_i - \bar{x_i})||||(x_j - \bar{x_j})||}|$, which takes values in the interval [0, 1].

Some of the most common distance metrics between boolean vectors are:

- **Hamming** distance: It is the number of positions in which the two boolean vectors are different.

- **Jaccard** distance: Jaccard distance between two Boolean vectors x_i, x_j is defined as $d(x_i, x_j) = \frac{C_{TF} + C_{FT}}{C_{TT} + C_{TF} + C_{FT}}$ with C_{ab} the number of places k for which $x_i[k] = a$ and $x_j[k] = b$ and a, b take values T, F.

- **Dice** distance: Dice distance between two Boolean vectors x_i, x_j is defined as $d(x_i, x_j) = \frac{C_{TF} + C_{FT}}{2C_{TT} + C_{TF} + C_{FT}}$ with C_{ab} the number of places k for which $x_i[k] = a$ and $x_j[k] = b$ and a, b take values T, F. We can calculate the Jaccard distance from the Dice distance and vice versa. The Dice distance does not satisfy the triangular inequality.

1.4 Clustering Evaluation Metrics

One of the most dominant and non-trivial issues is the evaluation of the clustering quality achieved by an algorithm. There are two categories of such indicators according to whether they use external information to evaluate the data clustering.

1.4.1 Internal Clustering Evaluation Metrics

Clustering evaluation metrics that do not use any external information about how we expect the data to get clustered are called internal. In general, these methods evaluate the clustering in terms of how compact are the resulting clusters and how well separated are the different clusters.

There can be used simple statistics such as within-group variance or the sum of squared error of the group members regarding the group center. However, more complex metrics have also been developed, with the following being commonly used:

- **Calinski – Harabasz**: This index is defined as $s = \frac{trace(B)}{trace(W)} \times \frac{N-k}{k-1}$, where N is the number of examples, k is the number of groups, $B = \sum_{j=1}^{k} \sum_{x_i \varepsilon S_j} (x_i - c_j)(x_i - c_j)^\intercal$ with c_j the center of the group j and $W = \sum_{j=1}^{k} |\Sigma_j|(c_j - c)(c_j - c)^\intercal$ with c the center of all data and $|\Sigma_j|$ the number of members of group j. Also, it is called variance ratio and takes values greater than 0. The higher this ratio is, the more compact and well-separated the groups are [10].

- **Davies – Bouldin**: This index is defined as $s = \frac{1}{k}\sum_{i=1}^{k} \max_{j \neq i} (\frac{d_i + d_j}{d_{ij}})$, where d_i is the average distance of the members of group i from its center and d_{ij} the distance between the centers of groups i and j. It expresses the average similarity shown within groups and takes values greater than 0 with a lower value indicating better clustering [10].

- **Dunn**: This index is defined as $s = \frac{\min_{i \neq j, i,j=1\cdots k} \delta_{ij}}{\max_{i=1\cdots k} \Delta_i}$ with δ_{ij} the distance between groups i, j and Δ_i the diameter of group i. There are many variations for the calculation of these sizes, with the most common being the distance of two groups to be considered as the minimum of the distances of the members of one group to the members of the other, and as the diameter of a group, the maximum of the distances between its members. A higher index value indicates better clustering [10].

1.4.2 External Clustering Evaluation Metrics

External evaluation metrics require additional information to evaluate clustering. In particular, they need the actual partitioning of the data into groups and compare it with the one achieved by the algorithm. In the following, we will consider Y_{true} as the actual partitioning of the data and Y as that achieved by the evaluated algorithm. Some of the most common external evaluation metrics are:

- **F-measure**: It is defined as $F = \frac{a}{\frac{1}{2}(c+d)}$ with a the number of examples that belong to the same group in both partition Y_{true} as in Y, c the number of examples that belong to the same group in partition Y_{true} and different ones in Y and d is the number of examples which belong to different groups in the partition Y_{true} and the same group in Y. This index takes values in the interval [0, 1]. A value of 1 indicates a partition that achieves perfect precision and recall.

- **Purity**: We consider that each group of Y corresponds to the group of Y_{true} to which most members of Y belong. The purity metric is defined as the percentage of examples assigned to the correct group according to the previous assignment and takes values in the interval [0, 1]. A value of 1 indicates perfect clustering.
- **Jaccard** and **Dice**: Jaccard and Dice distances, as defined above, can also be used to calculate how similar Y and Y_{true} are.

1.5 Artificial Neural Networks

In recent years, artificial neural networks (ANNs) have been used with great success to solve problems that machine learning deals with. An ANN is a structured collection of simple computing units called neurons. An ANN is organized in layers. It consists of an input layer, one or more hidden layers, and an output layer. Usually, the neurons of one layer are interconnected with all the neurons of the next layer and only with them.

Each neuron is provided with an input vector x and characterized by a vector of weights w. The two vectors have the same dimensions. It calculates the quantity $u = (wx + w_0)$, where w_0 is a number called neuron's bias. The quantity u is called neuron's stimulation. Usually, as weights of a neuron, we consider not only its weights and bias but also the extended weight vector $W = (w_0, w)$. So, when we use as an input vector $x = (1, x)$, neuron's stimulation is calculated as $u = Wx$. Finally, the output of the neuron $y = f(u)$ is calculated by applying an activation function to the stimulation u. The output of the neuron is provided as input to the neurons of the next level. In general, we want the activation function to be non-linear so that the network can approximate complex functions by making non-linear transformations on its input.

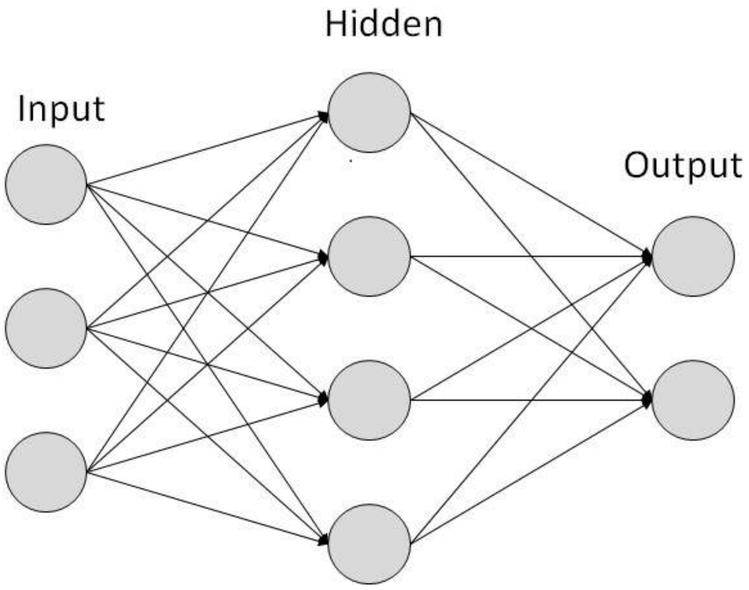

Figure 1.1: This figure shows the schema of a general Neural Network.

The most common activation functions are [11]:

- The **identity** function: $f(x) = x$. It is generally only used at the entry level because it's linear. Also, its derivative is constant and independent of x.
- The **step** function: $f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$. It is generally not used because its derivative is 0 and independent of x. Therefore, it makes it difficult to apply optimization methods to update the values of the weights.
- The **sigmoid** or **logistic** function: $\sigma(x) = \frac{1}{1+e^{-x}}$. It behaves similarly to the step function, taking values in the interval [0, 1]. However, it is differentiable with a derivative depending on x. Thus, there could be used optimization methods, but its derivatives often take very small values. As a result, the weights' values do not change enough and the convergence slows down during the training of the network.
- The **hyperbolic tangent (tanh)** function: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. It has similar behavior to the sigmoid function and the same problem with small values in the derivatives. Finally, it takes values in the interval [−1, 1].
- The **Rectified Linear Unit (ReLU)** function: $f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$. It is differentiable with derivatives that depend on x for $x > 0$ and does not have the

problem of small derivative values. So it is a widely used activation function. Its only problem is that for $x \leq 0$, the derivatives are 0, so it creates neurons that can be permanently inactive.

- The **Leaky ReLU** function: $f(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases}$. A commonly used value for parameter a is 0.01. This function tries to solve the problem of derivatives for $x \leq 0$ that ReLU function has.

- The **Softplus** function: $f(x) = \ln(1 + e^x)$. It could be considered as a smoother version of ReLU. Also, while it has no problems with its derivatives, because it is a more complex function, it is not often used.

Moreover, the functions that are commonly used at the output level are:

- The **Softmax** function: $f(x_i) = \frac{e^{x_i}}{\sum\limits_{j \neq i} e^{x_j}}$, where i is a neuron and j are all the remaining neurons in the layer. This function shows us a probability distribution regarding the neurons of the output layer, giving a greater probability to the neurons with the largest output. It allows us to use the output layer to classify the data, assuming that each neuron in the output layer represents some category.

- The **Softmin** function: $f(x_i) = \frac{e^{-x_i}}{\sum\limits_{j \neq i} e^{-x_j}}$. It is similar to Softmax but gives a higher probability to the neurons with the smallest output.

Training of an ANN is the determination of the values of neurons' weights and biases to minimize the value of a loss function, i.e. a function that expresses the performance of the network.

1.5.1 Backpropagation

The backpropagation algorithm is commonly used in the training of an ANN to efficiently calculate the derivatives of the loss function for the neurons' weights and biases. Specifically, let a network with L levels, input x, W^l the table with the weights of level l, where w_{ij}^l is the weight of edge from neuron j of level $l-1$ to neuron i of level l, f^l the activation function of neurons of level l and $L(x)$ the loss function.

Initially, network's output is calculated as $y = f^l(W^l f^{l-1}(W^{l-1} f^{l-2}(\cdots W^2 f^1(W^1 x))))$ and from this, the value of loss function $L(y)$. We define the derivative of the loss function over the stimulation of level $l(u^l)$ as $\delta^l = \frac{\partial L}{\partial u^l}$. δ^l can be regarded as the loss at level l. The quantities δ^l could be easily calculated, starting from level l and going backward till level 1.

At level L, the calculation is done through the relation $\delta^L = f^{L'}(u^L)\nabla_y L$ with $\nabla_y L$ the derivative of loss function over the network's output. In the rest levels, we use the recursive formula $\delta^L = f^{l'}(u^L)W^{L+1}\delta^{l+1}$. Having calculated these quantities, it is easy to calculate the derivatives of loss function over the neurons' weights as $\frac{\partial L}{\partial w^l} = \delta^l f^{l-1}(u^{l-1})$.

1.5.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is the most common method of finding the values of weights that minimize the value of the loss function. According to this method, we iteratively update the values of the weights with respect to the derivatives of the loss function over them, following the steps below:

1. Initialize the weights to random values.

2. Split the data into small groups of fixed size called batches.

3. Choose the groups one by one. Insert each example of the group into the network and calculate the derivatives of the loss function over the weights for each example through the backpropagation algorithm. After all the examples of a group have been inserted into the network, update the weights according to the relation $W^{new} = W^{old} - \eta \sum\limits_{x_i \varepsilon group} \frac{\partial L}{\partial w}$. The parameter η is called learning rate and regulates how much the values of the weights will change at each update. After this process for each group is completed, we say that an epoch has passed.

4. Repeat step 3 until the value of the loss function converges or repeat it for a certain number of epochs.

We have a serial update if each batch contains only one instance. If we have only one batch that includes all data, then we have the classic version of gradient descent. It has been found that SGD with a relatively small batch size usually leads to faster convergence than the classic one.

1.5.3 Adam

Adam is another commonly used algorithm for finding the values of the weights that minimize the value of the loss function. It follows the same logic as that of SGD but differs in how it updates the weights' values. Specifically, it follows the steps below to update weights [12]:

1. Defines the exponential moving average of the derivatives m. It initializes this average to 0 and updates it after each instance as $m^{new} = \beta_1 m^{old} + (1 - \beta_1)\frac{\partial L}{\partial w}$.
2. Defines the exponential moving average of the squares of derivatives u. It initializes this average to 0 and updates it after each instance as $u^{new} = \beta_2 u^{old} + (1 - \beta_2)(\frac{\partial L}{\partial w})^2$.
3. The above two averages are tendentious towards 0 due to their initialization. This is solved by calculating the non-tendentious averages as $\hat{m} = \frac{m}{1-\beta_1^t}$ and $\hat{u} = \frac{u}{1-\beta_2^t}$ with t the number of updates that have become in mean values.
4. Update the weights according to the relation $W^{new} = W^{old} - \eta\frac{\hat{m}}{\sqrt{\hat{u}+\varepsilon}}$

The parameters β_1 and β_2 control the rate at which the averages change. ε is a constant for numerical stability reasons.

1.5.4 LVQ

LVQ algorithm is a competitive learning algorithm that can be used to cluster data by finding the centers of k groups and assigning each example to the group with the closest center to it [13]. Training of the algorithm is to find the k centers. It becomes according to the following steps:

1. Initialize the centers with some of the known methods as described in k-means.
2. For each example x_i find the winner-center c_v, i.e. the center with the shortest distance from the example. Refresh only the winning center according to the relation $c_v^{new} = c_v^{old} + \eta(x_i - c_v^{old})$, where η is the learning rate.
3. Repeat this process until the centers converge.

The implementation of this algorithm can be done efficiently with a neural network consisting of two layers. An input layer that is fully interconnected with an output layer. The input layer consists of as many neurons as the data features, while the output layer consists of k neurons representing the centers of the k groups. Each example is provided to the network, and the weights of the winner-neuron (c_v) are updated according to the above steps.

1.6 Thesis Contribution

This thesis considers a novel expansion of the silhouette score [14], designated as the "soft silhouette score" [15]. This score assesses the effectiveness of probabilistic clustering solutions directly, thereby obviating the necessity for conversion into discrete forms. In addition to this evident benefit, a noteworthy aspect of the soft silhouette score is its differentiability with respect to the probabilities ascribed to cluster assignments. In the event that such probabilities are provided by a parametric machine learning model, the soft silhouette score has been employed as a clustering objective function for training parametric probabilistic models through the use of conventional gradient-based methods. In order to achieve this objective, a novel deep clustering approach based on autoencoders (AEs) has been proposed. This methodology offers cluster assignment probabilities directly through the network outputs and utilizes the soft silhouette score as a clustering objective [15]. The training of the network with the soft silhouette score allows for the achievement of two objectives: the minimization of inter-cluster variance and the maximization of the margin between clusters in the embedded space. We conducted an extensive evaluation of the deep clustering method. The experimental results indicate that the utilization of the Soft Silhouette loss function can facilitate the development of deep neural networks (DNNs) capable of addressing a diverse array of clustering problems.

Chapter 2

Deep Clustering

2.1 Introduction

Deep Learning (DL) is a powerful tool that can extract complex and valuable data representations from large datasets, reducing the need for extensive human-crafted features [16, 17]. A substantial proportion of DNN architectures place significant emphasis on an initial unsupervised learning phase, commonly referred to as unsupervised pretraining. AEs represent a prominent example of this approach. The acquisition of more refined and profound representations or features of the underlying data is enabled by this methodology, which has been widely acknowledged for substantially enhancing the outcomes of subsequent supervised learning tasks.

Although initially applied primarily to supervised learning tasks, DL has recently demonstrated considerable success in a range of unsupervised learning domains, including clustering and DR. One of the principal advantages of utilizing DL for clustering is its capacity to harness the representational capabilities of DNNs for pre-processing clustering inputs, thereby improving the quality of clustering outcomes. The utilization of DNNs in this manner enables the effective transformation of data into spaces that are more conducive to clustering. This transformation exploits the intrinsic capability of DNNs to perform highly nonlinear operations, enabling them to discern intricate patterns and structures within the data. This differs from traditional clustering methods, such as k-means or spectral clustering, which frequently rely on raw or linearly transformed data. Nevertheless, these methods may prove inadequate when dealing with datasets exhibiting intricate statistical properties.

In essence, integrating DL techniques into clustering processes enables the creation of more adaptive and flexible representations of the data, thereby improving clustering performance, particularly in scenarios involving complex and high-dimensional datasets.

Even though clustering has not historically been a primary focus of DL, the richness and robustness of the deep representations it provides make it a suitable choice for application in the field of clustering [18, 19]. For the sake of simplicity, the term "Deep Clustering" will be used throughout this thesis to refer to clustering methods involving DL. Deep clustering can be defined as a set of clustering techniques that employ DNNs to learn representations conducive to clustering, as discussed in [19].

2.2 Deep Neural Network Architectures

In the majority of deep clustering methods, the primary function of the DNN is to encode the inputs into a latent representation that is subsequently utilized for clustering purposes. In previous studies, a variety of DNN architectures have been utilized to achieve this objective [20].

- **Multilayer Perceptron (MLP)**: This type of feedforward network, described in [21], typically consists of at least one hidden layer of neurons without a linear activation function. In this architecture, the output of each layer serves as the input to the next layer, creating a sequential flow of information through the network.

- **AEs**: represent a specialized category of algorithms that are adept at learning efficient data representations without the necessity of labeled data. These ANNs are designed for unsupervised learning, with the objective of compressing and accurately representing input data. The fundamental principle of an AE is based on its dual-component structure, comprising an encoder and a decoder. The encoder transforms the input data into a condensed, lower-dimensional representation, which is often termed the “latent space” or “encoding”. Subsequently, the decoder reconstructs the original input from this representation. Through this process of encoding and decoding, the network discerns meaningful patterns within the data, facilitating the extraction of essential features [22].

  Architecture of AE in DL

  The general architecture of an AE comprises three principal layers: an encoder, a decoder, and a bottleneck layer.

  1. **Encoder**: In an AE, the encoder is the portion of the network responsible for transforming the raw input data into a compact and informative representation. Its architecture typically consists of multiple hidden layers, which progressively reduce the dimensionality of the input while capturing crucial features and patterns. These hidden layers collectively form the encoder.

     The bottleneck layer, also known as the latent space, is situated at the core of the encoder. This final hidden layer markedly reduces the dimensionality of the input data, effectively compressing it into a condensed encoding. The latent space serves as a compressed representation of the input data, capturing its essential characteristics in a lower-dimensional form.

  2. **Decoder**: The bottleneck layer receives the encoded representation and expands it back to match the dimensionality of the original input. This layer plays a pivotal role in ensuring that the compressed encoding is translated into a format consistent with the original data’s dimensions.

     Subsequently, the hidden layers assume the responsibility of incrementally augmenting the dimensionality of the data, thereby striving to reconstruct the original input. Their objective is to meticulously refine the encoded representation, gradually expanding it to encompass the intricacies and nuances of the input data.

The ultimate objective of the output layer is to generate a reconstructed output that closely resembles the input data. In an ideal scenario, this output would mirror the input data as closely as possible, thereby demonstrating the success of the AE in faithfully reproducing the original data from its compressed representation.

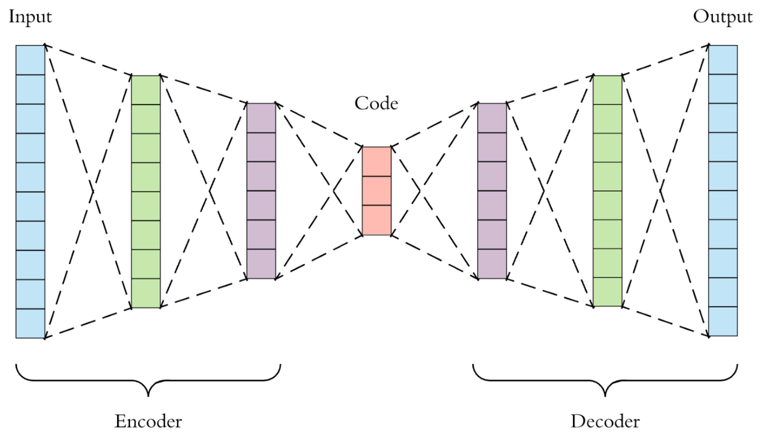

Figure 2.1: This figure shows the schema of a general AE.

Types of AEs

AEs are classified into various types, each of which possesses distinctive advantages and disadvantages.

1. **Denoising AE**: Denoising AEs operate on partially corrupted inputs with the objective of reconstructing the original undistorted data. This approach effectively precludes the network from merely replicating the input, instead prompting it to discern the intrinsic structure and salient characteristics of the data.

   Advantages:

   (a) **Feature Extraction**: Denoising AEs are particularly adept at extracting crucial features from data sets while simultaneously reducing the influence of noise or irrelevant features. By focusing on the restoration of the original, unadulterated input, the network is able to discern and prioritize the most significant characteristics.

   (b) **Data Augmentation**: They can be employed as a means of data augmentation, whereby additional training samples are generated from the restored images. This facilitates an improvement in the model's generalization ability and enhances its performance.

Disadvantages:

(a) **Noise Selection**: The determination of the optimal type and level of noise to introduce can be a challenging process, and it may require the input of domain experts. The selection of an appropriate noise source can have a significant impact on the efficacy of the denoising process.

(b) **Information Loss**: It is possible that the denoising process may result in the inadvertent loss of information that is crucial for accurate reconstruction. Such a loss may affect the fidelity of the output and, consequently, the overall performance of the AE.

2. **Sparse AE**: This specific type of AE typically comprises a greater number of hidden units than the input, yet only a limited subset is permitted to be active at any given time. This property is known as the sparsity of the network. The degree of sparsity can be adjusted through a number of methods, including manually setting specific hidden units to 0, modifying the activation functions employed, or incorporating a dedicated loss term into the cost function during training. These strategies facilitate the precise control of the network's sparsity, thereby enabling the adjustment of its behavior and the learned representations.

   Advantages:

   (a) **Noise and Irrelevant Feature Filtering**: The sparsity constraint inherent in sparse AEs serves to filter out noise and irrelevant features during the encoding process. By prompting only a limited number of neurons to be active, the AE prioritizes the capture of essential information, while minimizing the influence of extraneous details.

   (b) **Learning Important Features**: Sparse AEs frequently demonstrate proficiency in discerning and retaining significant and pertinent features, a capability that can be attributed to their emphasis on sparse activations. This prioritization of key features enhances the model's capacity to represent the underlying structure of the data in an efficient manner.

   Disadvantages:

   (a) **Hyperparameter Sensitivity**: The efficacy of sparse AEs is contingent upon the appropriate selection of hyperparameters. To achieve optimal results, it is essential to conduct careful tuning, as different inputs should ideally result in the activation of different nodes in the network. Insufficiently optimal parameter settings may result in suboptimal performance or ineffective feature extraction.

   (b) **Increased Computational Complexity**: The incorporation of sparsity constraints results in a notable increase in the computational complexity of sparse AEs. The introduction of sparsity constraints during the training phase introduces an additional computational burden, which may impact the overall training time and the required resources. This enhanced complexity may present challenges, particularly in circumstances where computational resources are constrained or when dealing with extensive datasets.

3. **Variational AE**: This approach is predicated on certain assumptions regarding the distribution of latent variables and employs the Stochastic Gradient Variational Bayes estimator during the training process. It operates under the assumption that the data is generated by a directed graphical model and aims to learn an approximation to $q_\phi(z|x)$, representing the conditional property $q_\theta(z|x)$ where ϕ and θ denote the parameters of the encoder and the decoder, respectively.

   Advantages:

   (a) **Data Generation**: Variational AEs (VAEs) are highly effective at generating new data points that closely resemble the original training data. The generation of these samples is based on the latent space, thus enabling the model to produce novel yet realistic data instances.

   (b) **Probabilistic Framework**: VAEs provide a probabilistic framework for the learning of compressed representations of data, thereby capturing both the underlying structure and the variations present within the data set. This capability has proven invaluable for tasks such as anomaly detection and data exploration, where an understanding of the data's inherent variability is essential.

   Disadvantages:

(a) **Approximation Error**: VAEs are based on approximations that are used to estimate the true distribution of latent variables. The approximation inherent within this process introduces a degree of error, which may impact the quality of the generated samples and the fidelity of the learned representations.

(b) **Limited Coverage of Data Distribution**: The samples generated by VAEs may only span a restricted subset of the true data distribution. This limitation may result in a lack of diversity in the generated samples, which could potentially limit the model's ability to capture the full range of variability present in the data.

4. **Convolutional AE**: Convolutional AEs utilize convolutional neural networks (CNNs) as their fundamental architectural component. The encoder component comprises multiple layers that accept image or grid inputs and process them through a series of convolutional layers. This process results in the generation of a compressed representation of the input data. In contrast, the decoder component performs the inverse operation of the encoder. The original image is then reconstructed by deconvolving the compressed representation, which is achieved through the use of transposed convolutional layers. The symmetrical design of convolutional AEs allows for the effective capture of spatial relationships in the data, resulting in accurate reconstructions.

   Advantages:

   (a) **Efficient Compression**: Convolutional AEs are particularly adept at compressing high-dimensional image data into a lower-dimensional format. This results in enhanced storage efficiency and facilitates the transmission of image data, particularly in scenarios where bandwidth or storage capacity is limited.

   (b) **Robust Reconstruction**: Convolutional AEs are capable of reconstructing missing components of an image, rendering them particularly well-suited for applications such as image inpainting. Furthermore, they are capable of handling images with slight variations in object position or orientation, thereby ensuring robust performance across diverse datasets.

Disadvantages:

(a) **Overfitting Risk**: Convolutional AEs are prone to overfitting, particularly when utilizing complex datasets. It is of vital importance to implement appropriate regularization techniques in order to mitigate this risk and guarantee that the model is capable of generalizing effectively to data that has not been previously encountered.

(b) **Data Loss and Quality Degradation**: The compression of data inherent in convolutional AEs has the potential to result in data loss, which may in turn lead to the reconstruction of images of a lower quality. It is essential to carefully consider the trade-off between compression efficiency and reconstruction fidelity, particularly in applications where the preservation of image quality is of paramount importance.

- **Deep Belief Network (DBN)**: is a probabilistic generative graphical model, designed with the purpose of extracting intricate hierarchical representations from input data [23]. It comprises numerous layers of stochastic latent variables. These variables progressively uncover deeper insights into the underlying structure of the data. The DBN architecture is constituted by the stacking of multiple Restricted Boltzmann Machines (RBM) [24]. Each RBM is composed of two distinct layers: a visible layer and a hidden layer. The distinctive configuration entails the utilization of the concealed layer of each RBM as the visible layer for the subsequent RBM in the sequence. This interconnected structure enables the learning of complex features through successive layers [20].

- **Radial Basis Function (RBF) Networks**: are frequently employed in the context of ANNs for the task of approximating functions. These networks are distinguished by their universal approximation capability and faster learning rates in comparison to other neural network architectures [25]. In the context of network architecture, which is typically composed of three layers, the initial layer represents the input nodes, which are followed by a hidden layer containing several non-linear activation units based on RBFs. The final layer represents the network's output. In the majority of cases, Gaussian functions are employed as the activation functions in RBF networks [26].

  To illustrate the working flow of the RBF network, suppose we have a dataset D which has N patterns of (x, y) where x is the input of the dataset and y is

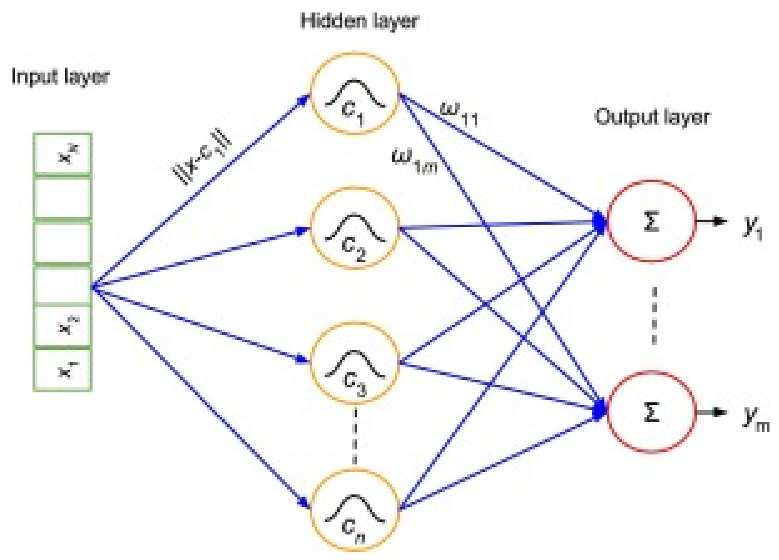

Figure 2.2: This figure shows the schema of a general RBF network.

the actual output. The activation of the ith function ϕ_i in the hidden layer of the network is computed using Eq. 2.1, which relies on the distance between the input pattern x and the center c_i:

$$\phi_i(\|x - c_i\|) = \exp\left(-\frac{\|x - c_i\|^2}{2\sigma_i^2}\right). \tag{2.1}$$

Here, $\|\cdot\|$ represents the Euclidean norm, c_i and σ_i are the center and width of the ith hidden neuron, respectively.

Then, the output of the kth node in the output layer of the network is calculated using:

$$y_k = \sum_{i=1}^{N} w_{ki}\phi_i(x). \tag{2.2}$$

The majority of classical approaches to training RBF networks in the literature entail two stages. In the initial stage, the centers and widths are determined, frequently through the utilization of unsupervised clustering algorithms. In the subsequent stage, the connection weights between the hidden layer and the output layer are adjusted with a view to minimizing an error criterion, such as the Mean Squared Error (MSE), across the entire dataset.

The training process for the RBF model is terminated either when the calculated error reaches a predetermined threshold (e.g., 0.01) or after a specified number of training iterations (e.g., 500). A specific number of nodes, typically 10, are selected for the hidden layer, and a Gaussian function is commonly employed as the transfer function in the computational units. In practice, RBF networks often demonstrate a more rapid convergence during training in comparison to MLP networks [25].

2.3 Deep Features

The DNN architecture allows for the utilization of deep features for clustering purposes, which can be sourced from one or multiple layers of the DNN:

- **Single layer**: In this configuration, a single layer of the DNN is employed for the extraction of deep features. This approach is typically advantageous due to its lower dimensionality.
- **Multiple layers**: This scenario entails the extraction of the deep features from a fusion of outputs originating from multiple layers. Consequently, the representation becomes more intricate, thereby enabling the embedded space to depict more complex semantic representations. This augmented capacity frequently results in enhanced outcomes in similarity computation [27].

2.4 Non-Clustering Loss

The non-clustering loss component is concerned exclusively with the learning of a deep representation of the data through the application of DL techniques. It operates independently of the clustering aspect of the process. A variety of non-clustering loss functions are available for consideration [20]:

- **Absence of non-clustering loss**: In this scenario, the network model is constrained only by the clustering loss. The absence of a non-clustering loss may result in the generation of inferior representations or even the collapse of clusters [6].
- **Reconstruction loss**: In the case of an AE being selected as the DNN architectural choice, the non-clustering loss is equivalent to the reconstruction loss.

Typically, it is defined as the distance measure $d_{\mathrm{AE}}(x_i, \hat{x}_i)$ between the input x_i to the AE and its corresponding reconstruction $\hat{x}_i = f(x_i)$. The most common formulation of this loss is the MSE between the two variables:

$$\mathcal{L}(x_i, f(x_i)) = d_{\mathrm{AE}}(x_i, f(x_i)) = \sum_{i=1}^{n} ||x_i - f(x_i)||^2, \tag{2.3}$$

where n is the number of data points, x_i represents the input, and $f(x_i)$ denotes the AE reconstruction.

- **Implicit maximum likelihood loss**: The objective of this loss function is to address the challenges associated with direct likelihood maximization, particularly when utilizing a DNN as the underlying machine learning model [28]. The implicit maximum likelihood loss is expressed as follows:

$$\mathcal{L}(x_i, x_{\mathrm{e}\theta_i}) = \sum_{i=1}^{n} ||x_i - x_{\mathrm{e}\theta_i}||_2^2. \tag{2.4}$$

  Here, n represents the number of data points, x_i denotes the i-th data point, and $x_{\mathrm{e}\theta_i}$ signifies the nearest sample (from m samples generated by the DNN model) to the i-th data point, determined using a distance metric (typically the Euclidean distance).

- **The min-max loss**: In the case of a generative adversarial network (GAN) being the optimal model, the min-max loss function is utilized [29]. This non-clustering loss function is defined as:

$$\mathcal{L}(D, G) = \min_{G} \max_{D} V(D, G) = \mathbb{E}_{x \sim p_{\mathrm{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]. \tag{2.5}$$

- **Additional Information**: Non-clustering loss can leverage additional data information to extract valuable features, even if this extra information isn't conducive to the clustering process.

2.5 Clustering Loss

The purpose of clustering loss functions is to direct neural networks towards the acquisition of representations of the input data that are conducive to the formation of clusters. Consequently, these functions are designated as clustering loss functions [18, 20, 19]. These can be classified into two categories: principal clustering loss and auxiliary clustering loss [20].

- **Principal clustering loss**: Following DNN training with this category of clustering loss, the clusters can be obtained directly. This implies that such clustering loss functions encompass both the cluster centroids and the process of data clustering. Examples of these types of clustering losses include k-means loss [30], cluster assignment hardening loss [31], agglomerative clustering loss [32], cluster classification loss [33], non parametric maximum margin clustering [34], and so forth.

- **Auxiliary clustering loss**: This category of clustering loss encourages the DNN to learn a representation that is more suitable for clustering. However, it does not directly provide clustering solutions. Consequently, deep clustering methods that employ auxiliary clustering loss require the execution of a clustering method after DNN training in order to obtain the clusters. Examples of auxiliary clustering losses deployed in deep clustering include the locality-preserving loss [35], which ensures that the DNN preserves the local properties of data embedding, and the group sparsity loss [35], which utilizes a block diagonal similarity matrix for representation learning.

2.5.1 Principal Clustering Loss

The following section presents a selection of principal clustering loss functions [19]:

- **No clustering loss**: Although a DNN trained exclusively with non-clustering losses can still generate deep features suitable for clustering post-training, it transforms the input data into a lower-dimensional representation, effectively reducing the dimensionality. While this transformation may occasionally prove beneficial for clustering, the incorporation of a clustering loss has been demonstrated to yield superior results [6, 31].

- k-**means loss**: The k-means loss, also referred to as clustering error, is designed to promote the generation of a representation that is conducive to clustering, as outlined in [6]. By minimizing this error with respect to the DNN parameters, the distance between each data point and its assigned cluster center is reduced, thus improving the quality of the clustering when applying k-means.

- **Agglomerative clustering loss**: In the context of agglomerative clustering, the process involves the progressive combination of two clusters with the highest affinity or similarity at each iteration until a predefined stopping criterion is met, as outlined by [36].

- **Cluster classification loss**: During the updating of clusters, the assignments may be employed as alternative class labels for a classification loss integrated into an auxiliary network branch. This configuration encourages the extraction of meaningful features across all layers of the network, as evidenced by the findings of [33, 37].

2.5.2 Auxiliary Clustering Loss

The subsequent section enumerates a selection of potential auxiliary clustering loss functions [19]:

- **Locality-preserving loss**: The objective of the locality-preserving loss function is to maintain the proximity of clusters by encouraging nearby data points to be positioned in close proximity to each other [35]. The mathematical expression is as follows:
$$\mathcal{L}_{lp} = \sum_{i} \sum_{j \in N_k(i)} s(x_i, x_j) ||f(x_i) - f(x_j)||^2. \tag{2.6}$$
Here, $N_k(i)$ denotes the set of k nearest neighbors of the data point x_i, $s(x_i, x_j)$ represents a similarity measure between the points x_i and x_j, and $f(\cdot)$ represents the nonlinear transformation performed by a DNN.

- **Group sparsity loss**: It draws inspiration from spectral clustering, employing a block diagonal similarity matrix for representation learning, as outlined by [38]. Group sparsity represents an effective technique for the selection of features. For example, in [35], the hidden units were divided into G groups, where G represents the assumed number of clusters. Given a data point x_i, the resulting representation takes the form $f_g(x_i)_{g=1}^{G}$. Accordingly, the loss function can be formulated as follows:
$$\mathcal{L}_{gs} = \sum_{i=1}^{N} \sum_{g=1}^{G} \lambda_g ||f_g(x_i)||. \tag{2.7}$$

Here, λ_g represents the weights assigned to sparsity groups, defined as $\lambda_g = \lambda\sqrt{n_g}$, where n_g denotes the group size and λ is a constant.

2.6 Synthesizing the Loss Functions

Let us consider a deep clustering process that employs both a clustering and a non-clustering loss function. It is of great importance to effectively combine these two losses. One common approach to achieve this is through the following formulation:

$$\mathcal{L}(\theta) = \alpha\mathcal{L}_c(\theta) + (1 - \alpha)\mathcal{L}_n(\theta), \tag{2.8}$$

where $\mathcal{L}_c(\theta)$ represents the clustering loss, $\mathcal{L}_n(\theta)$ represents the non-clustering loss, and $\alpha \in [0, 1]$ is a constant determining the balance between the two loss functions. α is employed as an additional hyperparameter during the training of DNN, and its value may be modified at various points throughout the training process in accordance with a predefined schedule. The scheduling methods for adjusting α during training are described in detail in [19]:

- **Joint training**: α remains constant within the range $0 < \alpha < 1$, and the DNN training incorporates both loss functions simultaneously.

- **Variable schedule**: The value of α undergoes a change throughout the training process in alignment with a predefined schedule. To illustrate, it may commence at a relatively low level and then undergo a gradual increase with each training epoch.

- **Pretraining, fine-tuning**: In the initial stage, the parameter α is set to 0, and the DNN undergoes training using only the non-clustering loss. Subsequently, in the second stage, α is set to 1, and the DNN is retrained solely using the clustering loss. This approach enables the DNN to be fine-tuned specifically for clustering tasks. However, training the DNN exclusively with the clustering error for an extended period may result in suboptimal clustering outcomes.

2.7 Updating Clusters

Clustering methods are typically divided into two main categories: hierarchical and partitional (centroid-based) approaches [2]. In hierarchical clustering, a hierarchy of clusters and data points is constructed. In contrast, partitional clustering involves the organization of data into clusters with distinct centers. This is achieved through the application of metric relations, which assign each data point to the cluster with the most similar center [20].

In the field of deep clustering, two prominent methodologies have been widely employed. The first is Agglomerative clustering integrated with DL [36], which represents a hierarchical clustering approach. The second is k-means coupled with DL [31, 6, 33, 39], which falls under the category of partitional clustering methods.

In the context of DNN training, a centroid-based approach entails the updating of both the clusters and the centroids. The prevailing methodologies for updating these elements are delineated as follows [20]:

- **Simultaneously updated with the network model**: The incorporation of cluster assignments as probabilities permits their representation as network parameters, thereby facilitating optimization through backpropagation.

- **Updated alternately with the network model**: The process of adjusting clustering assignments is conducted independently of the network model update procedure, as outlined in [31, 36].

2.8 Autoencoders As DNNs In Clustering

In recent years, DNNs have seen an increase in applications within the domain of DL, particularly in the area of clustering. DNNs are adept at learning intricate and valuable data representations from datasets, reducing the reliance on extensively engineered features by humans [40]. This versatility is owed to their exceptional nonlinear mapping capability and adaptability [16, 41]. Although clustering was not initially a primary focus of DL, numerous methods have emerged that leverage the representational power of neural networks, leading to the emergence of a new category of deep clustering methods. These approaches aim to enhance clustering outcomes by effectively training neural networks to transform input data and produce representations conducive to clustering. The ideal outcome is the formation of compact and well-separated clusters in the latent space [20, 19, 18, 42, 43].

In the context of deep clustering, a range of DNN models have been utilized, as evidenced in [43]. It is noteworthy that the most commonly used architectures include GANs [29], VAEs [44], and Graph Neural Networks [45]. Moreover, conventional DNNs have been integrated into methodologies like ClusterGan [37], VaDE [46], JULE [36], and NIMLC [47, 48].

Nevertheless, the majority of methodologies employ AEs, which stand out as the most prevalent DL model for clustering. AE-based deep clustering approaches seek to capitalize on the nonlinear capabilities of the encoder and decoder models to facilitate the shaping of the latent space [6]. To achieve this goal, novel objective functions have been proposed that combine the traditional AE reconstruction error with a clustering loss. This combination enables the training of the AE network in such a way that the data forms more compact clusters in the learned embedded space. This is achieved by simultaneously minimizing a clustering objective and the AE reconstruction error, thus ensuring that the original data information is preserved while the clustering objective is met.

Chapter 3

Autoencoder-Based Deep Clustering Methods

3.1 Dimensionality Reduction and Clustering

Many conventional learning methods traditionally treat DR and clustering as separate processes. However, recent studies have revealed that optimizing these tasks jointly can significantly enhance the performance of both. This novel approach operates under the assumption that data samples stem from a linear transformation of latent representations conducive to clustering [6].

In practice, the application of DR preprocessing techniques, such as Principal Component Analysis (PCA) or Non-negative Matrix Factorization (NMF) [49, 50], to reduce the dimensionality of x_i to a lower-dimensional space frequently results in enhanced outcomes when followed by k-means clustering. In addition to employing DR as a preprocessing step, the literature has also explored the approach of jointly performing DR and clustering [51, 52, 53].

Let's now consider a generative model where a data sample is represented as $x_i = Wh_i$, with $W \in \mathbb{R}^{M\times R}$ and $h_i \in \mathbb{R}^R$, where $R \ll M$. It is assumed that the data clusters are well-separated in the latent domain (i.e., where h_i resides), but the transformation introduced by W may result in distortion. In their work, Yang et al. (2017) [53] presented a joint optimization problem formulated as follows:

$$\min_{M,\{s_i\},W,H} \|X - WH\|_F^2 + \lambda \sum_{i=1}^{N} \|h_i - Ms_i\|_2^2 + r_1(H) + r_2(W) \tag{3.1}$$

s.t. $s_{j,i} \in \{0, 1\}$ and $\mathbf{1}^\top s_i = 1 \ \forall i, j$, where $X = [x_1, \ldots, x_N]$, $H = [h_1, \ldots, h_N]$, and $\lambda \geq 0$ serves as a parameter balancing the competing constraints of data fidelity and latent cluster structure. This formulation addresses both DR and latent clustering, with the regularization terms $r_1(\cdot)$ and $r_2(\cdot)$ serving to prevent trivial solutions. The data model $X \approx WH$ as presented in the previously referenced work, may be overly simplified. It is possible that the data generating process is more complex than a simple linear transformation, and therefore, further investigation is required. Therefore, there is justification in seeking powerful non-linear transformations, such as DNNs, to model this data-generating process. Concurrently, the joint DR and clustering approach remains a valuable technique.

Furthermore, the approach proposed by Xie et al. (2016) [31] and Yang et al. (2016) [36] involves connecting a clustering module to the output layer of a DNN and jointly learning the parameters of the DNN and the clusters. In particular, the methodologies seek to address an optimization problem of the following form:

$$\min_{W,\Theta} \hat{\mathcal{L}} = \sum_{i=1}^{N} q(f(x_i; W); \Theta), \tag{3.2}$$

where $f(x_i; W)$ represents the network output for the data sample x_i, W collects the network parameters, and Θ denotes parameters of some clustering model. For example, Θ represents the centroids M and the assignments $\{s_i\}$ when the k-means clustering formulation 1.1 is used. The function $q(\cdot)$ in the equation 3.2 represents some clustering loss, such as the Kullback-Leibler (KL) divergence loss in [31] or the agglomerative clustering loss in [36].

The best possible solution to the problem 3.2 occurs when $f(x_i; W) = 0$, and achieving an optimal objective value of $\hat{\mathcal{L}} = 0$ is always possible. Another kind of trivial solution is to group random data points tightly, resulting in a low $\hat{\mathcal{L}}$ value. However, this approach may not be ideal, since it doesn't take into account the individual data samples x_i's.

3.2 Deep Clustering Network

The study by Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos and Mingyi Hong [6], proposes a joint DR and k-means clustering method that uses DNNs for DR. By utilizing a DNN, they aim to retain the benefits of jointly optimizing both tasks while taking advantage of the DNN's ability to approximate complex nonlinear functions. As a result, this approach is adaptable to a wide range of generative models.

To achieve this goal, they elaboratively designed the structure of the DNN and formulated an optimization criterion that includes both DR and clustering objectives. Moreover, a practical and scalable algorithm tailored to effectively handle the optimization problem is introduced. Finally, this clustering method is called Deep Clustering Network (DCN).

3.2.1 Framework

The DCN framework integrates DR and k-means clustering, where DR is performed by using a DNN that deviates from conventional linear models. The goal of their study is to improve the modeling of the data transformation process with a more comprehensive model, which can result in a latent space that is much more compatible with k-means clustering. Thus, they introduce an optimization criterion for the joint DNN-based DR and k-means clustering, which consists of three components: DR, data reconstruction, and cluster structure-promoting regularization. Lastly, they present a comprehensive solution package that includes empirically effective initialization methods and a novel alternating stochastic gradient algorithm.

3.2.2 Formulation

The objective of this work was to develop a model that could effectively represent the relationship between the observable data x_i and its clustering-friendly latent representation h_i. This was achieved by employing a nonlinear mapping, specifically, $h_i = f(x_i; W)$, where $f(\cdot; W) : \mathbb{R}^M \to \mathbb{R}^R$ denotes the mapping function and W denotes the set of parameters. In this instance, the mapping function is a DNN, given that DNNs possess the capacity to approximate any continuous mapping using a finite number of parameters.

Nevertheless, the primary issue that arises in this methodology is the manner by which trivial solutions can be avoided. The primary means of preventing them in linear DR lies in the reconstruction phase, particularly in the term $\|X - WH\|_F^2$ within equation 3.1. This component ensures that the learned h_i's can effectively reconstruct the x_i's with the help of W.

Moreover, to prevent trivial low-dimensional representations such as all-zero vectors, DCN employs a decoding network $g(\cdot; Z)$ to map the h_i's back to the data domain. Additionally, it necessitates that $g(h_i; Z)$ and x_i exhibit a high degree of similarity under a specified metric, such as mutual information.

So, they built the below cost function:

$$\min_{W,Z,M,\{s_i\}} \sum_{i=1}^{N} \left(l\left(g\left(f(x_i)\right), x_i\right) + \frac{\lambda}{2}\|f(x_i) - Ms_i\|_2^2 \right), \tag{3.3}$$

s.t. $s_{j,i} \in \{0, 1\}, \quad \mathbf{1}^T s_i = 1 \quad \forall i, j$, where $f(x_i; W) = f(x_i)$ and $g(h_i; Z) = g(h_i)$, respectively.

The function $l(\cdot) : \mathbb{R}^M \to \mathbb{R}$ represents a specific loss function that measures the reconstruction error. In this study, the least squares loss function is employed, defined as $l(x, y) = \|x - y\|_2^2$. The parameter $\lambda \geq 0$ is a regularization parameter that balances the trade off between the reconstruction error and the generation of k-means friendly latent representations.

3.2.3 Optimization Procedure

This section presents the optimization strategy employed to enhance the performance and efficiency of the clustering algorithm. The objective of this procedure is to achieve an optimal balance between accurate feature representation and optimal clustering results. This is accomplished by combining pretraining, alternate optimization, and iterative parameter updates.

Layer-wise Pretraining for Initialization

The authors propose to use the layer-wise pretraining method, as introduced by Bengio et al. (2007) [54] for training AEs, to initialize the network parameters (W, Z). While this pretraining technique may be unnecessary for large-scale supervised learning tasks, the authors highlight its significance for their proposed DCN, which operates exclusively in an unsupervised manner. Regardless of the dataset size, the layer-wise pretraining procedure has been demonstrated to be crucial. Following the pretraining phase, the outputs of the bottleneck layer are subjected to k-means clustering, thereby providing the initial values for M and $\{s_i\}$.

Alternating Stochastic Optimization

Even with good initialization, solving the problem 3.3 remains a significant challenge. The commonly used SGD algorithm cannot be directly applied to the joint optimization of W, Z, M, and $\{s_i\}$ because the block variable $\{s_i\}$ is restricted to a discrete set. The idea is to to integrate the principles of alternating optimization and SGD. In particular, the approach entails optimizing the subproblems with respect to one of M, $\{s_i\}$, and (W, Z), while keeping the other two sets of variables fixed.

- **Update Network Parameters**

  For fixed $(M, \{s_i\})$, the subproblem w.r.t. (W, Z) is analogous to training a sparse AE, but with an additional penalty term on the clustering performance. It makes use of the well-established tools for training DNNs, e.g., backpropagation based SGD and its variants. To implement SGD for updating the network parameters, the problem is considered w.r.t. the incoming data point x_i:

  $$\min_{W,Z} \mathcal{L}^i = \ell(g(f(x_i)), x_i) + \frac{\lambda}{2}\|f(x_i) - Ms_i\|_2^2. \tag{3.4}$$

  The gradient of the above function with regard to the network parameters is easily computable, i.e., $\nabla_X \mathcal{L}^i = \frac{\partial \ell(g(f(x_i)),x_i)}{\partial X} + \lambda \frac{\partial f(x_i)}{\partial X}(f(x_i) - Ms_i)$, where $X = (W, Z)$ is a collection of the network parameters. The gradients $\frac{\partial \ell}{\partial X}$ and $\frac{\partial f(x_i)}{\partial X}$ can be calculated by backpropagation (Rumelhart et al., 1988) [55] (it should be noted that the aforementioned derivatives are actually subgradients with respect to X, given that the ReLU function is not differentiable at 0). Subsequently, the network parameters are updated as follows:

  $$X \leftarrow X - \alpha \nabla_X \mathcal{L}^i, \tag{3.5}$$

  where $\alpha > 0$ is a diminishing learning rate.

- **Update Clustering Parameters**

  In the context of fixed network parameters and M, the assignment vector of the current sample, i.e., s_i, can be updated in an online fashion in a natural manner. Specifically, the update of s_i is conducted as follows:

$$s_{j,i} = \begin{cases} 1, & \text{if } j = \arg\min_{k=\{1,\dots,K\}} \|f(x_i) - m_k\|_2, \\ 0, & \text{otherwise.} \end{cases} \tag{3.6}$$

When fixing $\{s_i\}$ and X, the update of M is simple and may be done in a variety of ways. For example, one may simply employ the formula $m_k = \frac{1}{|C_k^i|} \sum_{i \in C_k^i} f(x_i)$, where C_k^i is the recorded index set of samples assigned to cluster k from the first sample to the current sample i. Although the previously described update is intuitive, it could be problematic for online algorithms. This is because the historical data that has already been recorded (i.e., $x_1, \dots, x_i$) may not be sufficiently representative enough to model the global cluster structure. Additionally, the initial s_i's values may be significantly inaccurate.

Therefore, simply averaging the current assigned samples may cause numerical problems. Instead of doing the above, they employ the idea in (Sculley, 2010) [56] to adaptively modify the learning rate associated with the updating of $m_1, \dots, m_K$. The underlying concept is straightforward: assume that the clusters are approximately balanced in terms of the number of data samples they contain. Then, following the updating of M for a number of samples, it is recommended that the centroids of the clusters with a considerable number of assigned members be updated in a more gradual manner, while those with a lesser number of assigned members be updated in a more pronounced manner, in order to maintain balance.

To implement this, let c_k^i be defined as the count of the number of times the algorithm assigned a sample to cluster k before handling the incoming sample x_i. Then, m_k can be updated by a simple gradient step:

$$m_k \leftarrow m_k - \frac{1}{c_k^i}(m_k - f(x_i))s_{k,i}, \tag{3.7}$$

where the gradient step size $\frac{1}{c_k^i}$ controls the learning rate. The previously described update of M can also be regarded as an SGD step, resulting in an overall alternating block SGD procedure that is summarized in Algorithm 3.1. It should be noted that an epoch represents a single pass of all data samples through the network.

Algorithm 3.1 Alternating Stochastic Gradient Descent (SGD)

```
Require: Initialization {Perform $T$ epochs over the data}.
1: for $t = 1 : T$ do
2:     Update network parameters by 3.5.
3:     Update assignment by 3.6.
4:     Update centroids by 3.7.
5: end for
```

Algorithm 3.1 exhibits a number of advantageous characteristics. Firstly, the algorithm can be implemented in a completely online fashion, thereby ensuring scalability. Secondly, numerous established techniques for optimizing the performance of DNN training can be directly applied, including the mini-batch version of SGD and batch-normalization [57].

3.3 Deep Embedded Clustering

In their study, Junyuan Xie, Ross Girshick, and Ali Farhadi propose an unsupervised clustering method for jointly learning feature representations and cluster assignments [31]. This clustering algorithm is designated as Deep Embedded Clustering (DEC). DEC draws inspiration from recent advancements in DL, particularly in computer vision tasks, where DNNs have demonstrated success in learning more effective features [16, 58, 59, 60].

However, in contrast to supervised learning methodologies, DEC is designed to facilitate unsupervised clustering. It proposes a parameterized non-linear mapping from the original data space to a lower-dimensional feature space, optimized based on a clustering objective.

3.3.1 Framework

A number of variants of the k-means algorithm have been proposed, including joint DR and clustering methods, which address issues associated with higher-dimensional input spaces [61, 62]. However, these methods are constrained to linear embedding, whereas the DEC approach employs DNNs for non-linear embedding, which is well-suited to complex data.

Furthermore, the discussion includes an examination of spectral clustering and its variants [63], which offer more flexible distance metrics and have been demonstrated to outperform k-means in a number of cases. Techniques that combine spectral clustering and embedding are explored in [64, 65], but they often suffer from scalability issues due to the necessity of computing the full graph Laplacian matrix.

To address concerns regarding scalability, approximate algorithms for spectral clustering have been developed; nevertheless, these algorithms may compromise performance in favor of speed. In contrast, the DEC method exhibits linear growth in relation to the number of data points and is designed to handle large datasets in a manner that is both efficient and effective.

Additionally, the excerpt discusses the application of KL divergence for data visualization and DR, with techniques such as t-SNE [66, 67]. Thus, drawing inspiration from parametric t-SNE, the DEC method establishes a centroid-based probability distribution and endeavors to minimize its KL divergence from an auxiliary target distribution. This approach aims to enhance both the precision of clustering assignments and the fidelity of feature representations.

In order to address the challenges inherent in clustering when there is a lack of labeled data, the DEC framework introduces an iterative refinement process. In particular, it utilizes an auxiliary target distribution derived from the current soft cluster assignments to incrementally enhance both clustering accuracy and feature representation quality. In conclusion, the approach represents a promising avenue for unsupervised clustering, as it enables the simultaneous learning of feature representations and cluster assignments through the use of DL techniques.

3.3.2 Formulation

Consider the task of clustering a set of n points $\{x_i \in \mathbb{X}\}_{i=1}^{n}$ into k clusters, each of which is characterized by a centroid μ_j, where $j = 1, \dots, k$. In contrast to the conventional approach of directly clustering within the original data space $\mathbb{X}$, DEC proposes an initial step of transforming the data through a nonlinear mapping, specifically $f_\theta : \mathbb{X} \to \mathbb{Z}$, where θ denotes the learnable parameters and $\mathbb{Z}$ represents the latent feature space. Typically, the dimensionality of $\mathbb{Z}$ is significantly smaller than that of $\mathbb{X}$ [68]. DNNs are selected as the optimal choice for parameterizing f_θ due to their inherent capacity for function approximation [69] and their proven ability to learn meaningful features [40].

The presented algorithm, DEC, operates by concurrently learning a set of k cluster centers $\{\mu_j \in \mathbb{Z}\}_{j=1}^{k}$ within the feature space $\mathbb{Z}$ and the parameters θ governing the DNN responsible for mapping data points into $\mathbb{Z}$. DEC comprises two distinct phases: (1) initialization of parameters using a deep AE [70] and (2) parameter optimization, also known as clustering. In the optimization phase, the algorithm iterates between computing an auxiliary target distribution and minimizing the KL divergence to it.

3.3.3 Optimization Procedure

The study presented in this section proposes an unsupervised algorithm to enhance the clustering process, utilizing an initial estimate of the non-linear mapping f_θ and the initial cluster centroids, represented by the set of values $\{\mu_j\}_{j=1}^{k}$. The algorithm comprises two iterative steps. Initially, a soft assignment is computed between the embedded points and the cluster centroids. Subsequently, the deep mapping function, f_θ, is updated and the cluster centroids are refined by leveraging the current highly confident assignments through an auxiliary target distribution. This iterative process continues until a convergence criterion is met.

Soft Assignment

In accordance with the methodology proposed by van der Maaten and Hinton (2008) [66], the Student's t-distribution is utilized as a kernel to assess the degree of similarity between an embedded point z_i and the centroid μ_j:

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\nu)^{-(\nu+1)/2}}{\sum\limits_{j'}(1 + \|z_i - \mu_{j'}\|^2/\nu)^{-(\nu+1)/2}}. \tag{3.8}$$

In this context, $z_i = f_\theta(x_i) \in \mathbb{Z}$ represents the embedding of $x_i \in \mathbb{X}$, and ν represents the degrees of freedom associated with the Student's t-distribution. The quantity q_{ij} signifies the probability of assigning sample i to cluster j, indicating a soft assignment. In the absence of a validation set for cross-validation of the ν in the unsupervised setting, and given the redundancy of its learning, they set $\nu = 1$ [67].

KL Divergence Minimization

In DEC method, it is proposed the iterative refinement of clusters through the exploitation of their high-confidence assignments, facilitated by the use of an auxiliary target distribution. In detail, the model is trained by aligning the soft assignment with the target distribution. In order to achieve this, the objective is defined as the KL divergence loss between the soft assignments q_i and the auxiliary distribution p_i. This is expressed as follows:

$$\mathcal{L} = \mathrm{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{3.9}$$

The selection of target distributions P is of critical importance in determining the performance of DEC. A straightforward approach might entail setting each p_i to a delta distribution (pointing to the nearest centroid) for data points exceeding a confidence threshold and disregarding the remainder. However, since q_i represents soft assignments, it is more natural and versatile to optimize for softer probabilistic targets. More specifically, this method strives to achieve the following objectives with respect to the target distribution: (1) enhanced prediction accuracy, (2) greater focus on data points with higher confidence levels, and (3) normalization of loss contribution to prevent large clusters from distorting the hidden feature space.

In this work, p_i is computed by first squaring q_i and then normalizing by the frequency of occurrences within a given cluster:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}}, \tag{3.10}$$

where $f_j = \sum_i q_{ij}$ represents the soft cluster frequencies.

The DEC training strategy may be regarded as a form of self-training [71]. Similar to self-training, the process begins with an initial classifier and an unlabeled dataset, which are then labeled with the classifier to train on its own high-confidence predictions. Indeed, experimental evidence indicates that DEC enhances the initial estimate in each iteration by learning from high-confidence predictions, thereby improving low-confidence ones.

Optimization

The optimization process entails a joint optimization of the cluster centers $\{\mu_j\}$ and DNN parameters θ through the application of SGD with momentum. The gradients of $\mathcal{L}$ with respect to the feature-space embedding of each data point z_i and each cluster centroid μ_j are computed as follows:

$$\frac{\partial \mathcal{L}}{\partial z_i} = \frac{\nu+1}{\nu} \sum_j \left(1 + \frac{\|z_i - \mu_j\|^2}{\nu}\right)^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j), \tag{3.11}$$

$$\frac{\partial \mathcal{L}}{\partial \mu_j} = -\frac{\nu+1}{\nu}\sum_i \left(1+\frac{\|z_i-\mu_j\|^2}{\nu}\right)^{-1} \times (p_{ij}-q_{ij})(z_i-\mu_j). \tag{3.12}$$

Subsequently, the gradients $\frac{\partial \mathcal{L}}{\partial z_i}$ are propagated to the DNN and employed in standard backpropagation to calculate the parameter gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ of the DNN. With regard to the assignment of clusters, the process is terminated when the proportion of points that alter their cluster assignment between two consecutive iterations is less than the total percentage.

3.3.4 Parameter Initialization

The discussion will now turn to the initialization process of the DEC algorithm, which involves the configuration of the DNN parameters θ and the cluster centroids $\{\mu_j\}$. To initiate this process, a stacked AE (SAE) is employed. SAEs are known for their ability to consistently generate semantically meaningful and well-separated representations in real-world datasets [70, 72, 73]. The unsupervised representations obtained from the SAE facilitate the learning of clustering representations within DEC.

The SAE network is initialized layer by layer, with each layer comprising a denoising AE trained to reconstruct the output of the preceding layer after random corruption [70]. This denoising AE is defined as follows:

$$\begin{aligned}
\tilde{x} &\sim \text{Dropout}(x),\\
h &= g_1(W_1\tilde{x}+b_1),\\
\tilde{h} &\sim \text{Dropout}(h),\\
y &= g_2(W_2\tilde{h}+b_2).
\end{aligned}$$

In this context, Dropout$(\cdot)$ represents a stochastic mapping that randomly sets to 0 a portion of its input dimensions [74]. The functions g_1 and g_2 serve as activation functions for the encoding and decoding layers, respectively. The model parameters are represented by $\theta = \{W_1, b_1, W_2, b_2\}$. The objective of the training process is to minimize the least-squares loss $\|x-y\|^2$. Subsequent to the training of each layer, the output h is employed as input for training the following layer. ReLUs are utilized in all encoder/decoder pairs, with the exception of g_2 in the first pair (which handles input data that may include positive and negative values) and g_1 in the last pair (to ensure the preservation of complete information in the final data embedding [70]).

Once the layer-wise training phase has been completed, the encoder layers are concatenated, followed by the decoder layers in reverse layer-wise training order. This combination results in the formation of a multilayer deep AE, comprising a bottleneck coding layer at its core. Further refinement through fine-tuning is employed with the objective of minimizing the reconstruction loss. Following this process, the decoder layers are discarded, leaving only the encoder layers to serve as the initial mapping between the data space and the feature space.

Finally, for initializing the cluster centers, the data is passed through the initialized DNN to obtain embedded data points. Standard k-means clustering is then performed in the feature space $\mathbb{Z}$ to derive the initial centroids $\{\mu_j\}_{j=1}^{k}$.

3.4 Improved Deep Embedded Clustering

The DEC algorithm establishes an efficient objective through self-learning. This objective (clustering loss) is employed to concurrently adjust the parameters of the transformation network and the cluster centers. By incorporating cluster assignment into soft labels, the algorithm attains implicit coordination. However, it is not possible to rely on the clustering loss alone to guarantee the preservation of local structure. Consequently, the feature transformation may become misguided, which could potentially compromise the integrity of the embedded space.

To address this issue, the work of Xifeng Guo, Long Gao, Xinwang Liu, Jianping Yin [75] posits that both the guidance of clustering-oriented loss and the preservation of local structure are crucial for effective deep clustering. Taking inspiration from Peng et al. (2016) [76], they employ an under-complete AE to learn embedded features while preserving the local structure of the data distribution. This approach involved integrating the AE into the DEC framework, which enabled their framework to jointly conduct clustering and acquire representative features while maintaining local structure integrity. This clustering method is called Improved Deep Embedded Clustering (IDEC).

3.4.1 Framework

The IDEC algorithm is essentially a modified version of DEC, which integrates an under-complete AE to preserve local structure. It distinguishes itself from Yang et al. (2016) [36] through its simplicity, which eliminates the need for recurrence, and it outperforms DEC in terms of both clustering accuracy and feature representativeness. In the IDEC algorithm, a denoising AE is employed for pretraining, and an undercomplete AE is integrated into the DEC framework subsequent to initialization.

3.4.2 Formulation

The primary contribution of DEC [31] lies in its utilization of the clustering loss (or target distribution P). This method relies on the use of highly confident samples as a form of supervisory input, with the objective of encouraging a denser distribution of samples within each cluster. Nevertheless, there is no guarantee that samples situated at the boundaries will be correctly allocated to the corresponding cluster. To address this challenge, IDEC explicitly preserves the local structure of the data. As a result, the supervision provided by highly confident samples assists in guiding marginal samples towards their correct clusters.

Let's consider about a dataset X comprising n samples, where each sample x_i belongs to $\mathbb{R}^d$, with d denoting the dimensionality of the space. The number of clusters K is known a priori, and each cluster j is characterized by a center $\mu_j \in \mathbb{R}^d$. For each sample x_i, a cluster index $s_i \in \{1, 2, \ldots, K\}$ is assigned. Two mappings are defined: the first, $f_W : x_i \to z_i$, represents a nonlinear transformation mapping x_i to its embedded point z_i in a low-dimensional feature space; the second, $g_{W'} : z_i \to x'_i$, maps z_i back to its reconstructed sample x'_i.

The objective is to identify a suitable f_W that enhances the embedded points $\{z_i\}_{i=1}^{n}$ for the clustering task. Two principal elements are of paramount importance: the AE and the clustering loss. The AE is utilized to learn representations in an unsupervised manner, thereby facilitating the preservation of the data's intrinsic local structure. The clustering loss function refines the embedded space to achieve a well-dispersed distribution of embedded points. The objective is defined as

$$\mathcal{L} = \mathcal{L}_r + \gamma \mathcal{L}_c, \tag{3.13}$$

where $\mathcal{L}_r$ and $\mathcal{L}_c$ denote the reconstruction loss and clustering loss, respectively. In this context, the parameter $\gamma > 0$ serves as a coefficient controlling the degree of distortion in the embedded space. When $\gamma = 1$ and $\mathcal{L}_r \equiv 0$, equation 3.13 simplifies to the objective of DEC 3.9.

3.4.3 Optimization Procedure

This section delineates the methodology designed to improve DEC's clustering performance by effectively initializing the model, preserving local structure, and employing an optimized training strategy. This approach balances precise feature representation with robust clustering outcomes.

Clustering Loss and Initialization

The clustering loss which is used is already defined in 3.3.3. Additionally, in accordance with the recommendations proposed by Xie et al. (2016) [31], a stacked denoising AE undergoes pretraining prior to commencing the clustering procedure. Upon completion of the pretraining phase, the embedded points are deemed to be valid feature representations for the input samples. Subsequently, the cluster centers $\{\mu_j\}_{j=1}^{K}$ can be initialized by applying the k-means on $\{z_i = f_W(x_i)\}_{i=1}^{n}$.

Local Structure Preservation

It is possible that the embedded points may not be inherently suited to the clustering task. In response, DEC [31] removes the decoder and refines the encoder using the clustering loss $\mathcal{L}_c$. However, this refinement process may result in distortion of the embedded space, which could diminish the representativeness of embedded features and potentially compromise clustering performance. Accordingly, in the IDEC method, the decoder is maintained in its original form and the clustering loss is directly applied to the embedded space.

In order to guarantee the efficacy of the clustering loss, it is inadvisable to employ the stacked denoising AE for pretraining. This is because clustering should ideally be performed on features extracted from clean data, rather than on data corrupted by noise, as is the case with the denoising AE. Consequently, the noise is removed directly. As a result, the stacked denoising AE is transformed into an undercomplete AE. The reconstruction loss is quantified using the MSE:

$$\mathcal{L}_r = \sum_{i=1}^{n} \|x_i - g_{W'}(z_i)\|_2^2, \tag{3.14}$$

where $z_i = f_W(x_i)$ and f_W and $g_{W'}$ represent the encoder and decoder mappings, respectively.

As demonstrated in prior research by [76] and [77], AEs are capable of preserving the local structure of the data-generating distribution. In light of this observation, it seems unlikely that making minor adjustments to the embedded space using the clustering loss will result in corruption. It is therefore recommended that the coefficient γ remains below 1.

Optimization

The optimization of Equation 3.13 is conducted using mini-batch SGD in conjunction with backpropagation. This process entails updating three sets of parameters: the weights of the AE, the cluster centers, and the target distribution P.

- **Update AE's weights and cluster centers**

  In the initial stage, the weights of the AE and the cluster centers are updated while maintaining the target distribution P at a fixed value. This allows for the computation of the gradients of $\mathcal{L}_c$ with respect to both the embedded point z_i and the cluster center μ_j. The following equations illustrate this:

$$\frac{\partial \mathcal{L}_c}{\partial z_i} = 2\sum_{j=1}^{K}\left(\frac{1}{1+\|z_i-\mu_j\|^2}\right)(p_{ij}-q_{ij})(z_i-\mu_j), \tag{3.15}$$

$$\frac{\partial \mathcal{L}_c}{\partial \mu_j} = 2\sum_{i=1}^{n}\left(\frac{1}{1+\|z_i-\mu_j\|^2}\right)(q_{ij}-p_{ij})(z_i-\mu_j). \tag{3.16}$$

  Then, for a mini-batch comprising m samples and with a learning rate λ, the update for μ_j is as follows:

$$\mu_j = \mu_j - \frac{\lambda}{m}\sum_{i=1}^{m}\frac{\partial \mathcal{L}_c}{\partial \mu_j}. \tag{3.17}$$

  The weights of the decoder are adjusted by:

$$W' = W' - \frac{\lambda}{m}\sum_{i=1}^{m}\frac{\partial \mathcal{L}_r}{\partial W'}. \tag{3.18}$$

  The weights of the encoder are adjusted by:

$$W = W - \frac{\lambda}{m}\sum_{i=1}^{m}\left(\frac{\partial \mathcal{L}_r}{\partial W} + \gamma\frac{\partial \mathcal{L}_c}{\partial W}\right). \tag{3.19}$$

- **Update target distribution**

  Updating the target distribution P is crucial as it serves as the "ground truth" soft label, which is influenced by the predicted soft label. To ensure stability, it is imperative that the target distribution P not be updated at every iteration by using a mini-batch of samples. In contrast, the target distribution is updated with all embedded points every T iterations. When updating the target distribution, the label assigned to x_i is obtained through the following equation:

$$s_i = \arg\max_j q_{ij}, \tag{3.20}$$

  where q_{ij} is computed by Equation 3.8. The training process will be terminated when the percentage change in label assignments between two consecutive updates for the target distribution is less than a predefined threshold δ.

Algorithm 3.2 Improved Deep Embedded Clustering (IDEC) algorithm

```
Require: Input data: $X$; Number of clusters: $K$; Target distribution update interval:
    $T$; Stopping threshold: $\delta$; Maximum iterations: $MaxIter$.
Ensure: AE's weights $W$ and $W'$; Cluster centers $\mu$ and labels $s$.
 1: Initialize $\mu$, $W'$, and $W$ according to Section 3.4.3.
 2: for iter $\in \{0, 1, \ldots, MaxIter\}$ do
 3:   if iter%$T == 0$ then
 4:     Compute all embedded points $\{z_i = f_W(x_i)\}_{i=1}^n$.
 5:     Update $P$ using Eqs. 3.8, 3.10, and $\{z_i\}_{i=1}^n$.
 6:     Save last label assignment: $s_{\text{old}} = s$.
 7:     Compute new label assignments $s$ via Eq. 3.20.
 8:     if $\frac{sum(s_{\text{old}} \neq s)}{n} < \delta$ then
 9:       Stop training.
10:     end if
11:   end if
12:   Choose a batch of samples $S \in X$.
13:   Update $\mu$, $W'$, and $W$ via Eqs. 3.17, 3.18, and 3.19 on $S$.
14: end for
```

Algorithm 3.2 summarizes the complete IDEC procedure.

Chapter 4

Deep Clustering using Soft Silhouette

4.1 Silhouette

The silhouette score, first proposed by [14], is a metric used to assess the quality of clustering solutions. It presupposes that an effective clustering solution comprises clusters that are both compact and well-separated. Given a dataset $X = \{x_1, \ldots, x_N\}$, which has been partitioned into K clusters $C = \{C_1, \ldots, C_K\}$, where $d(x_i, x_j)$ denotes the distance between x_i and x_j, the silhouette score can then be computed as follows:

The initial step is to calculate the individual silhouette score $s(x_i)$ for each data point x_i. Firstly, the average distance $a(x_i)$ is calculated for each data point with respect to all other data points within its cluster C_I:

$$a(x_i) = \frac{1}{|C_I| - 1} \sum_{x_j \in C_I, i \neq j} d(x_i, x_j), \tag{4.1}$$

where $|C_I|$ represents the cardinality of cluster C_I with $|C_I| > 1$. The value of $a(x_i)$ serves to quantify the degree of fit exhibited by the data point x_i within its cluster. A low value of $a(x_i)$ indicates that x_i is similar to its cluster members, which suggests that the data point is correctly grouped. Conversely, a higher value of $a(x_i)$ implies that x_i is distant from its cluster members.

In order to calculate the silhouette score, it is necessary to compute the minimum average distance between a data point x_i and data points in other clusters, which is denoted as $b(x_i)$. This measure, $b(x_i)$, is expressed as follows:

$$b(x_i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{x_j \in C_J} d(x_i, x_j). \tag{4.2}$$

A higher value of $b(x_i)$ signifies that the data point x_i exhibits a notable divergence from the data points in other clusters, which is regarded as a beneficial attribute.

The silhouette score, $s(x_i)$, for a data point x_i is determined by considering two key factors: firstly, the requirement for a small within-cluster distance $a(x_i)$, and secondly, a large between-cluster distance $b(x_i)$. In mathematical terms, it is defined as:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max\{a(x_i), b(x_i)\}}. \tag{4.3}$$

It is crucial to acknowledge that $-1 \leq s(x_i) \leq 1$. A value closer to 1 indicates that x_i is well-clustered within its assigned group, whereas a value near -1 suggests that x_i might have been incorrectly labeled as it displays a greater degree of similarity with points in other clusters.

The overall silhouette score for the entire partition C of the dataset X is calculated by averaging the individual silhouette values:

$$S(X) = \frac{1}{N} \sum_{i=1}^{N} s(x_i). \tag{4.4}$$

The silhouette score, as proposed by [14], serves not only as an effective means for internal clustering evaluation but also establishes an intuitive clustering objective that favors compact and well-separated clusters. However, existing deep clustering objectives are primarily concerned with achieving compact clustering solutions without explicitly optimizing for cluster separability. To address this limitation, a probabilistic variant of the silhouette score, termed the soft silhouette, has been introduced [15]. This variant enables optimization for both compactness and cluster separability.

4.2 Soft Silhouette

The soft silhouette score [15] considered in this work, represents an extension of the conventional silhouette score. Rather than utilizing hard cluster assignments, it considers probabilistic cluster assignments. In this context, suppose we have a dataset $X = \{x_1, \ldots, x_N\}$ partitioned into K clusters $C = \{C_1, \ldots, C_K\}$, and let $d(x_i, x_j)$ express the distance between data points x_i and x_j. Furthermore, let $P_{C_I}(x_i)$ denote the probability that x_i belongs to cluster C_I. It follows naturally that $\sum\limits_{i=1}^{K} P_{C_I}(x_i) = 1$. If it is assumed that x_i belongs to cluster C_I, then we define:

- $a_{C_I}(x_i)$: the distance between the point x_i and the cluster C_I. This is essentially a weighted average (expected value) of the distances from x_i to all other points $x_j \in X$, where the weights are the probabilities $P_{C_I}(x_j)$ indicating the likelihood that x_j belongs to the cluster C_I.

$$a_{C_I}(x_i) = \frac{\sum\limits_{j=1}^{N} P_{C_I}(x_j) d(x_i, x_j)}{\sum\limits_{j=1, j \neq i}^{N} P_{C_I}(x_j)}. \tag{4.5}$$

- $b_{C_I}(x_i)$: The minimum value of the expected distance of x_i from the other clusters C_J, which differ from C_I.

$$b_{C_I}(x_i) = \min_{J \neq I} \frac{\sum\limits_{j=1}^{N} P_{C_J}(x_j) d(x_i, x_j)}{\sum\limits_{j=1, j \neq i}^{N} P_{C_J}(x_j)} = \min_{J \neq I} a_{C_J}(x_i). \tag{4.6}$$

- $s_{C_I}(x_i)$: The conditional silhouette value for x_i given that it belongs to cluster C_I.

$$s_{C_I}(x_i) = \frac{b_{C_I}(x_i) - a_{C_I}(x_i)}{\max\{a_{C_I}(x_i), b_{C_I}(x_i)\}}. \tag{4.7}$$

The soft silhouette value $sf(x_i)$ of data point x_i is calculated as the expected value of $s_{C_I}(x_i)$ with respect to its cluster assignment probabilities $P_{C_I}(x_i)$:

$$sf(x_i) = \sum_{I=1}^{K} P_{C_I}(x_i) s_{C_I}(x_i), \tag{4.8}$$

and the total soft silhouette score $Sf(X)$ of the given partition is computed by aggregating (or averaging) the individual scores $sf(x_i)$:

$$Sf(X) = \frac{1}{N} \sum_{i=1}^{N} sf(x_i). \tag{4.9}$$

It is noteworthy that when dealing with hard clustering, where cluster assignment probability vectors are essentially one-hot vectors, the equations for soft silhouette are analogous to those of typical silhouette calculations.

As can be seen from the aforementioned equations, there is a clear differentiability with respect to the cluster assignment probabilities in the soft silhouette score. Consequently, it can be utilized as a viable clustering objective function within a DL framework. The principal advantage of this objective is in its capacity to optimize both cluster compactness and separation in a simultaneous manner. The following section presents a deep clustering approach based on this concept.

4.3 The DCSS method: Deep Clustering using Soft Silhouette

This section presents the Deep Clustering using Soft Silhouette (DCSS) algorithm [15], which falls under the category of AE-based deep clustering methods that utilize soft silhouette as a clustering loss.

As previously outlined in chapter 3, a standard AE-based deep clustering methodology entails the utilization of an encoder network $z = f_w(x)$, where $f_w(\cdot) : \mathbb{R}^d \to \mathbb{R}^m$, providing latent representations (embeddings) z, and a decoder network $\hat{x} = g_\theta(z)$, where $g_\theta(\cdot) : \mathbb{R}^m \to \mathbb{R}^d$, reconstructing the outputs given the embeddings. These networks are trained to optimize a total loss, which is the sum of the reconstruction loss and the clustering loss: $\mathcal{L}_{AE} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{cl}$.

In the DCSS approach, the soft silhouette score is employed for the clustering loss, which requires the specification of cluster assignment probabilities $p(x) = (p_1(x), \ldots, p_K(x))$ for an input x. In order to satisfy this requirement, the AE-model is enhanced with an additional network $h_r(z)$, designated as the clustering network. This network receives as input the embedding of a data point x, $z = f_w(x)$, and produces as output the cluster assignment probabilities $p_j(x) = h_{rj}(x)$ for $j = 1, \ldots, K$. Thus, given the dataset $X = \{x_1, \ldots, x_N\}$, we first compute the embeddings $z_i = f_w(x_i)$. Subsequently, we specify the pairwise distances $d(z_i, z_j)$ and the cluster assignment probability vectors $p(x_i) = h_r(z_i)$, which are essential for the calculation of the soft silhouette score.

The depicted architectural model is presented in Figure 4.1. It is noteworthy that the clustering network operates in conjunction with the decoder network. Given an input vector x, the model yields the embedding vector z, the reconstruction vector $\hat{x}$, and the probability vector $p(x)$.

Following an investigation of a number of potential alternatives, an RBF model

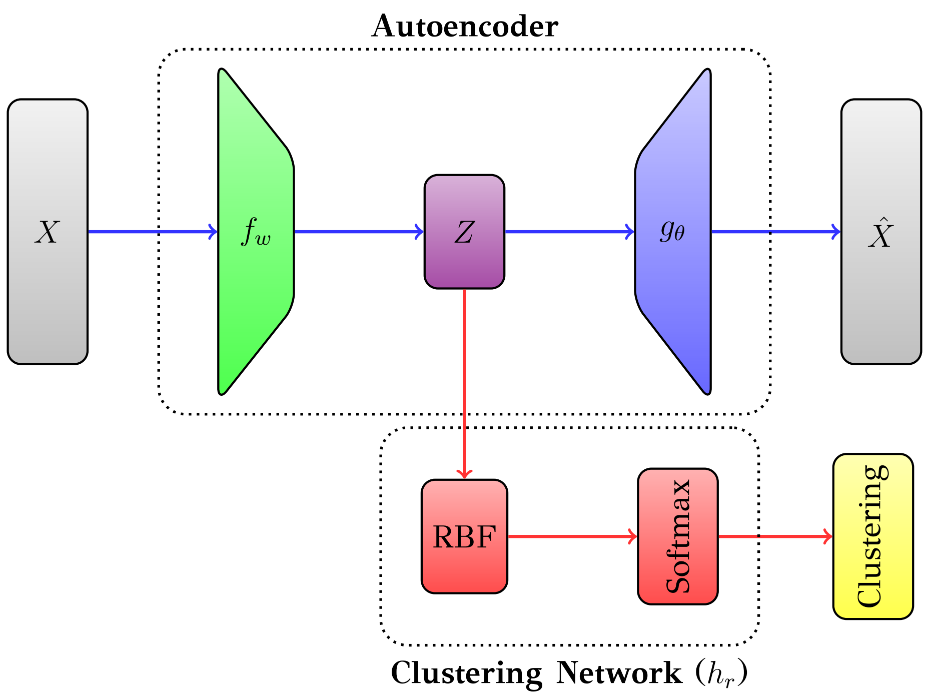

Figure 4.1: This figure shows the schema of the DCSS model architecture.

was selected as the clustering network, augmented with a softmax output unit to provide the probability vector for cluster assignments. The number of RBF units is set equal to the number of clusters K.

It is essential to maximize the soft silhouette criterion in order to obtain solutions of optimal quality. Given that a clustering loss is a quantity to be minimized, it is important to consider the constraint that $Sf \leq 1$. Therefore, the clustering loss is defined as:

$$\mathcal{L}_{cl} = 1 - Sf. \tag{4.10}$$

It is important to note that $\mathcal{L}_{cl}$ is always positive and attains its minimum value when Sf is at its maximum value ($Sf = 1$). Therefore, the total loss for model training is specified as follows:

$$\mathcal{L}_{AE} = \frac{1}{N}\sum_{i=1}^{N} ||x_i - g_\theta(f_w(x_i))||^2 + \lambda(1 - Sf(h_r(X))), \tag{4.11}$$

where $h_r(X) = \{h_r(x_1), \ldots, h_r(x_N)\}$ represents the cluster assignment probability vectors. It is crucial to highlight that the pairwise distances $d(f_w(x_i), f_w(x_j))$ between the embeddings are also integral to the computation of Sf. In this work, the Euclidean distance has been utilized. Since $\mathcal{L}_{AE}$ is differentiable with respect to the model parameters w, θ, and r, it can be minimized using gradient-based optimization procedures that are typical in this context.

A technical issue has been identified during the training of the model, whereby in a significant number of cases, the model converges to a trivial solution. This solution is characterized by output probabilities for many data points that tend to be uniform (i.e., equal to $\frac{1}{K}$). In order to address this issue, an additional term has been inserted to the objective function, which serves to penalize uniform solutions by minimizing the entropy of the output probability vectors. Accordingly, the entropy regularization term $\mathcal{L}_{reg}$ is defined as follows:

$$H(h_r(X)) = -\sum_{i=1}^{N}\sum_{j=1}^{K} h_{rj}(x_i)\log h_{rj}(x_i). \tag{4.12}$$

Algorithm 4.1 Deep Clustering using Soft Silhouette (DCSS) algorithm

```
Require: $X$ (dataset)
Require: $K$ (number of clusters)
Require: $\lambda_1, \lambda_2$ (regularization hyperparameters)
1: Randomly initialize the $w$ and $\theta$ AE parameters.
   Stage 1: Pretraining
2: Pretrain the encoder $f_w$ and decoder $g_\theta$ by minimizing the reconstruction error $\mathcal{L}_{rec}$ (eq. 3.14) through gradient-based optimization for $T_{pr}$ epochs. ▷ We employed batch training using the Adam optimizer.
3: Apply $k$-means with $K$ clusters to the learned representations $z = f_w(X)$.
4: Initialize the parameters of the clustering network $h_r$ using the $k$-means result.
   Stage 2: Training
5: Update the parameters $\theta$, $w$, and $r$ by minimizing the total loss (eq. 4.13) until convergence through gradient-based optimization to obtain $\theta^\star$, $w^\star$, and $r^\star$. ▷ We employed batch training using the Adam optimizer.
   Stage 3: Inference
6: Compute the clustering solution $C = \arg\max \text{softmax}(h_{r^\star}(f_{w^\star}(X)))$. ▷ Data clustering.
7: return the clustering solution $C$ and the learned parameters $w^\star$, $\theta^\star$, $r^\star$.
```

The final total loss function, which is minimized during the training of our model, is defined as:

$$\mathcal{L}_{AE} = \mathcal{L}_{rec} + \lambda_1\mathcal{L}_{cl} + \lambda_2\mathcal{L}_{reg}, \tag{4.13}$$

and in more detail:

$$\mathcal{L}_{AE} = \frac{1}{N}\sum_{i=1}^{N}||x_i - g_\theta(f_w(x_i))||^2 + \lambda_1(1 - Sf(h_r(X))) - \lambda_2\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K}h_{rj}(x_i)\log h_{rj}(x_i). \tag{4.14}$$

The particulars of the methodology, designated as DCSS, are delineated in Algorithm 4.1.

Chapter 5

Experiments

This chapter presents the results of our experiments conducted on real-world datasets. More specifically, the deep clustering performance of the DCSS approach is evaluated with a comparison to the most prevalent AE-based deep clustering methods discussed in chapter 3.

5.1 Datasets

Table 5.1 provides an overview of the benchmark datasets utilized for the experimental evaluation. The datasets exhibit a range of characteristics, including variations in size (n), dimensions (d), number of clusters (k), complexity, data type, and domain of origin. The following paragraphs provide a comprehensive overview of the datasets utilized in our study, along with the preprocessing steps applied to them.

The datasets employed in this work are Pendigits, EMNIST MNIST (EMNIST), and EMNIST Balanced Digits (BD). These datasets consist of handwritten digits categorized into ten classes, representing digits from 0 to 9. It is worth noting that the EMNIST dataset is an extended and more challenging version of the MNIST dataset [78]. Both the EMNIST and BD datasets consist of images with a resolution of 28 × 28 pixels. In contrast, the Pendigits data instances are represented by 16-dimensional vectors containing pixel coordinates. Additionally, the EMNIST Balanced Letters (BL) dataset was incorporated, comprising handwritten letters in both uppercase and lowercase forms, with a resolution of 28×28 pixels. The BL dataset is divided into three mutually exclusive subsets. The first subset includes the letters A to J, the second subset consists of the letters K to T, and the third subset contains the remaining letters, U to Z. The initial two subsets encompass 28,000 data points each, dispersed across 10 clusters. In contrast, the final subset comprises 16,800 data points and 6 clusters.

Moreover, we examined the Human Activity Recognition with Smartphones (HAR) dataset, which consists of data gathered from the accelerometer and gyroscope sensors of smartphones during various human activities. Each record in the dataset is a 560-feature vector with time and frequency domain variables. HAR comprises 6 classes of human activities: walking, walking upstairs, walking downstairs, sitting, standing, and laying.

Furthermore, the Waveform-v1 (WVF-v1) dataset was incorporated, consisting of 3 categories of generated waves with 5,000 data points each. Each class is generated from a combination of 2 of 3 “base” waves.

All datasets were subjected to a preprocessing step involving min-max normalization. This technique maps the attributes of each data point to the [0, 1] interval, thereby preventing attributes with large ranges from dominating the distance calculations and avoiding numerical instabilities in the computations [4].

5.2 Neural Network Architectures

The process of identifying optimal architectures and hyperparameters through cross-validation represents a significant challenge in the context of unsupervised learning scenarios. Therefore, we rely on commonly used architectures for the employed neural network models, thereby obviating the necessity for dataset-specific tuning. In the

Table 5.1: The datasets used in our experiments. N is the number of data instances, d is the dimensionality, and k denotes the number of clusters.

Dataset	**Type**	N	d	k	**Source**
EMNIST Balanced Digits	Image	28000	28×28	10	[79]
EMNIST MNIST	Image	70000	28×28	10	[79]
EMNIST Balanced Letters (A-J)	Image	28000	28×28	10	[79]
EMNIST Balanced Letters (K-T)	Image	28000	28×28	10	[79]
EMNIST Balanced Letters (U-Z)	Image	16800	28×28	6	[79]
HAR	Tabular	10299	560	6	[67]
Pendigits	Tabular	10992	16	10	[80]
Waveform-v1	Tabular	5000	21	3	[80]

case of tabular data, our approach entails the adoption of a well-established architectural framework consisting of fully connected layers [31, 67]. Specifically, the AE architecture we utilize is as follows:

$$x_d \to \mathrm{Fc}_{500} \to \mathrm{Fc}_{500} \to \mathrm{Fc}_{2000} \to \mathrm{Fc}_m \to \mathrm{Fc}_{2000} \to \mathrm{Fc}_{500} \to \mathrm{Fc}_{500} \to \hat{x}_d,$$

where Fc_m represents a fully connected layer with m neurons and x_d denotes a d-dimensional data vector.

In the context of image datasets, CNNs have been shown to be an effective tool for the capture of semantic visual features. Accordingly, we employ a convolutional-deconvolutional AE to learn embeddings for the image datasets. The AE architecture comprises three convolutional layers (encoder), one fully connected layer (embedding layer), and three deconvolutional layers (decoder) [81, 82, 83]. In particular, the architecture is structured as follows:

$$x_{28\times28} \to \mathrm{Conv}^5_{32} \to \mathrm{Conv}^5_{64} \to \mathrm{Conv}^3_{128} \to \mathrm{Fc}_m \to \mathrm{Deconv}^3_{128} \to \mathrm{Deconv}^5_{64} \to \mathrm{Deconv}^5_{32} \to \hat{x}_{28\times28},$$

where Conv^a_b (Deconv^a_b) represents a convolutional (deconvolutional) layer with an $a \times a$ kernel and b filters, with the stride always set to 2.

In the aforementioned encoder-decoder networks, the ReLU activation function is leveraged [84], with the exception of the embedded layer of the AE, where the Hyperbolic Tangent (tanh) function is applied. The He initialization method is employed for the initialization of weights and biases, as described in [85].

As previously stated, for the clustering network, we have selected a RBF model with the number of hidden units equivalent to the number of clusters. Subsequently, the outputs of the RBF units are then passed through a softmax activation function with K outputs, thereby providing the cluster assignment probabilities. Following the

AE pretraining phase, the centers of the RBF layer are initialized using the k-means algorithm in the embedded space, with the parameter σ set to a small positive value. Additionally, the temperature parameter T of the softmax function is fixed at $T = 20$.

5.3 Evaluation

It is crucial to note that, as clustering is an unsupervised task, all algorithms were designed in such a way that they were unaware of the true clustering of the data. In order for us to evaluate the performance of the clustering methods, we employed standard external evaluation measures, assuming the availability of ground truth clustering information [86]. In all cases, the number of clusters was set to the number of ground-truth categories, on the assumption that cluster labels and class labels are identical.

One of the evaluation measures employed is the Normalized Mutual Information (NMI) [87], which is defined as follows:

$$\mathrm{NMI}(Y, C) = \frac{2 \times I(Y, C)}{H(Y) + H(C)}, \tag{5.1}$$

where Y represents the ground-truth labels, C represents the cluster labels, $I(\cdot)$ represents the mutual information measure, and $H(\cdot)$ represents the entropy.

The second metric employed is the Adjusted Rand Index (ARI) [88, 89], which is a corrected version of the Rand Index [90]. ARI is a metric that quantifies the degree of overlap between two partitions and is defined as:

$$\mathrm{ARI}(Y, C) = \frac{RI(Y, C) - E[RI(Y, C)]}{\max\{RI(Y, C)\} - E[RI(Y, C)]}, \tag{5.2}$$

where $RI(\cdot)$ represents the Rand Index, and $E[\cdot]$ represents the expected value.

5.4 Experimental Setup

An exhaustive performance analysis of the studied DCSS method has been conducted, with a view to comparing it with well-established deep clustering methods, including DCN [6], DEC [31], and IDEC [75]. These methods are all designed to learn a cluster-friendly embedded space, similar to silhouette based deep clustering approach. Furthermore, the performance of the k-means algorithm was assessed in both the original data space and in the embedded space obtained from the AE (AE+k-means) [3]. A comparison of DCSS method with the latter approach allows us to quantify the performance enhancements achieved by incorporating an AE into the clustering process. It is crucial to acknowledge that, in order to guarantee a fair and accurate comparison between the deep clustering methods, we have utilized identical model architectures for all methods. This approach yielded more favorable clustering results than those reported in the original papers.

In experiments involving k-means, the algorithm was initialized 100 times, and the clustering solution with the lowest mean sum of squares error was selected. For the other methods that incorporate an AE in the clustering process, each experiment was repeated 10 times. In the case of deep clustering methods, an AE pretraining phase was conducted, and the clustering loss was disregarded. In the context of image datasets, the AE was subjected to a 100-epoch pretraining process with a learning rate of 1×10^{-3}. In contrast, for tabular datasets, the pretraining phase was extended to 1,000 epochs with a learning rate of 5×10^{-4}. A slight L_2 regularization of 1×10^{-5} was applied during the pretraining procedure. Thereafter, during the training phase, the deep clustering models were trained for 100 epochs with a learning rate of 5×10^{-4} and without any regularization penalty.

A fixed batch size of 256 was utilized, and the Adam optimizer [12] was employed with default settings of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ during both the pretraining and training phases. Moreover, a number of methodologies are available for adjusting the non-clustering loss (Eq. 4.11) in conjunction with the clustering loss (Eq. 4.10) [20] during the training phase. The most simplified and representative approach was selected by setting the hyperparameters to low values that achieve a balance between the two losses. In particular, we set $\lambda_1 = 0.01$ and $\lambda_2 = 0.01$. Finally, in order to initialize the centers of the RBF layer, we applied the k-means algorithm to the embeddings of the pretrained AE, while σ was initialized to a small positive value.

5.5 Results

This section presents the experimental results for each dataset. They are organized into two tables per dataset, with one table for each evaluation metric: NMI and ARI. Each row in these tables reports the mean, standard deviation, minimum, and maximum values of the respective metric for each examined deep clustering algorithm. Additionally, we include some figures with the optimal clustering results achieved by DCSS for each dataset. The t-SNE algorithm was applied to the latent data (output of the latent space) [66] to reduce its dimensionality to two dimensions, where it was necessary, in order to present the data in accordance with the clusters into which DCSS's RBF has grouped them. The data points within each cluster are represented by the same color.

5.5.1 EMNIST Balanced Digits

Table 5.2: Statistical Analysis of the NMI on the EMNIST Balanced Digits dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.48	0.00	0.48	0.48
AE + k-means	0.72	0.03	0.66	0.75
DCN	0.78	0.03	0.73	0.83
DEC	0.82	0.03	0.76	0.88
IDEC	0.84	0.02	0.80	0.87
DCSS	**0.87**	0.03	0.83	0.94

Table 5.3: Statistical Analysis of the ARI on the EMNIST Balanced Digits dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.36	0.00	0.36	0.36
AE + k-means	0.64	0.05	0.53	0.69
DCN	0.70	0.05	0.61	0.79
DEC	0.78	0.05	0.68	0.86
IDEC	0.78	0.04	0.71	0.84
DCSS	**0.82**	0.06	0.71	0.95

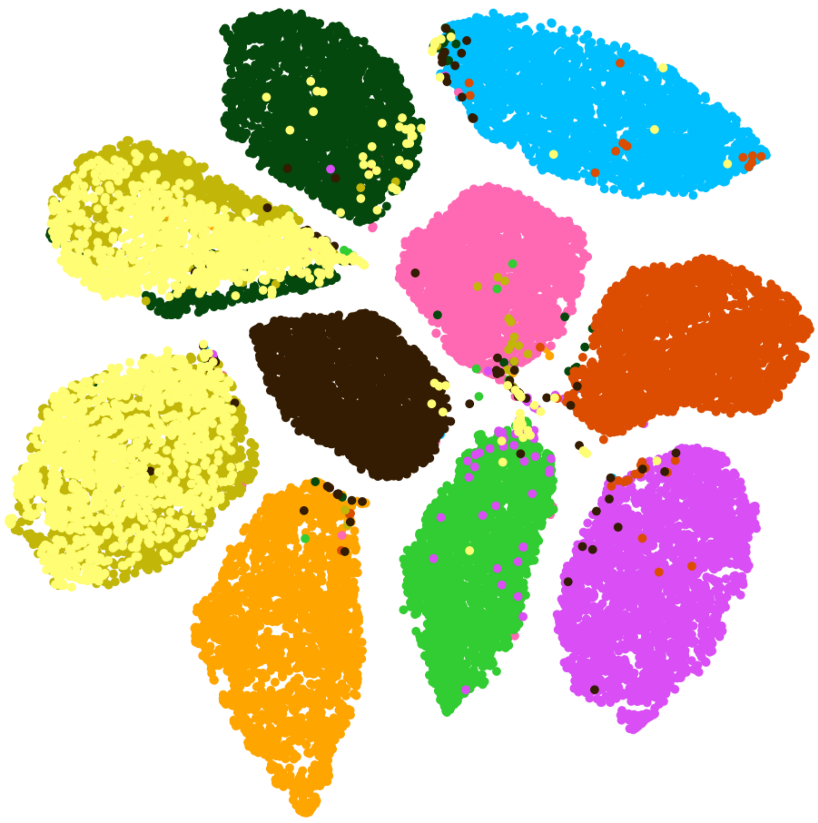

Figure 5.1: This figure illustrates the optimal clustering obtained by the DCSS algorithm on the EMNIST Balanced Digits dataset.

5.5.2 EMNIST MNIST

Table 5.4: Statistical Analysis of the NMI on the EMNIST MNIST dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.48	0.00	0.48	0.48
AE + k-means	0.76	0.01	0.73	0.78
DCN	0.83	0.02	0.80	0.83
DEC	0.85	0.03	0.80	0.90
IDEC	0.88	0.04	0.84	0.94
DCSS	**0.90**	0.02	0.87	0.91

Table 5.5: Statistical Analysis of the ARI on the EMNIST MNIST dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.35	0.00	0.35	0.35
AE + k-means	0.71	0.02	0.66	0.74
DCN	0.78	0.02	0.74	0.82
DEC	0.81	0.04	0.75	0.90
IDEC	**0.85**	0.06	0.79	0.95
DCSS	**0.85**	0.03	0.81	0.87

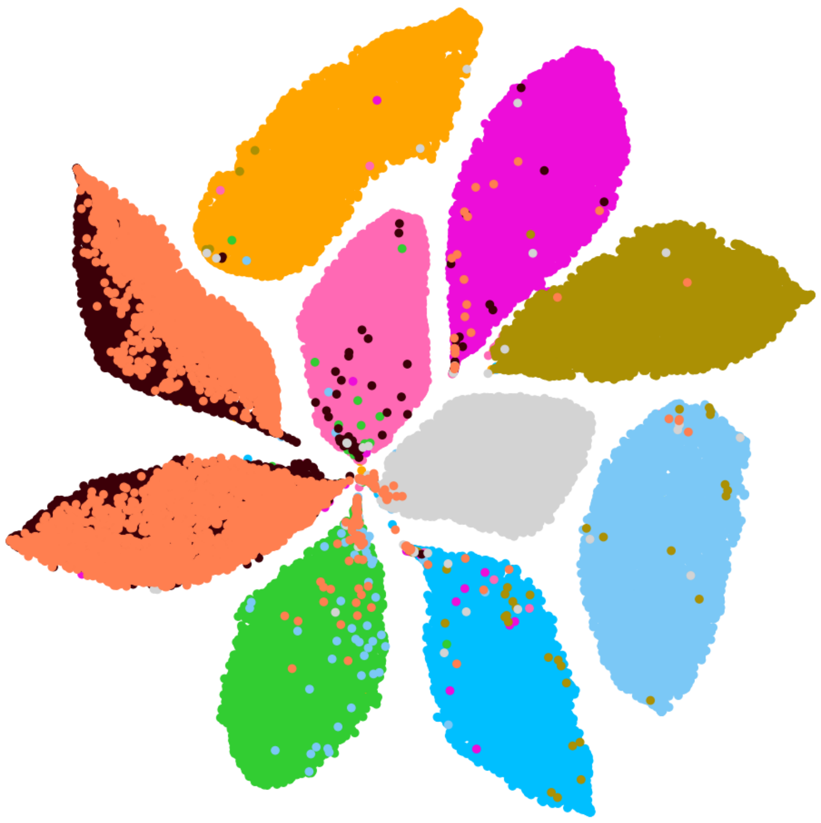

Figure 5.2: This figure illustrates the optimal clustering obtained by the DCSS algorithm on the EMNIST MNIST dataset.

5.5.3 EMNIST Balanced Letters (A-J)

Table 5.6: Statistical Analysis of the NMI on the EMNIST Balanced Letters (A-J) dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.35	0.00	0.35	0.36
AE + k-means	0.62	0.03	0.56	0.68
DCN	0.71	0.03	0.65	0.75
DEC	0.76	0.04	0.70	0.80
IDEC	0.74	0.03	0.69	0.80
DCSS	**0.79**	0.03	0.74	0.83

Table 5.7: Statistical Analysis of the ARI on the EMNIST Balanced Letters (A-J) dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.25	0.00	0.25	0.26
AE + k-means	0.50	0.04	0.43	0.57
DCN	0.61	0.05	0.50	0.66
DEC	0.68	0.05	0.62	0.75
IDEC	0.65	0.06	0.56	0.74
DCSS	**0.69**	0.06	0.59	0.77

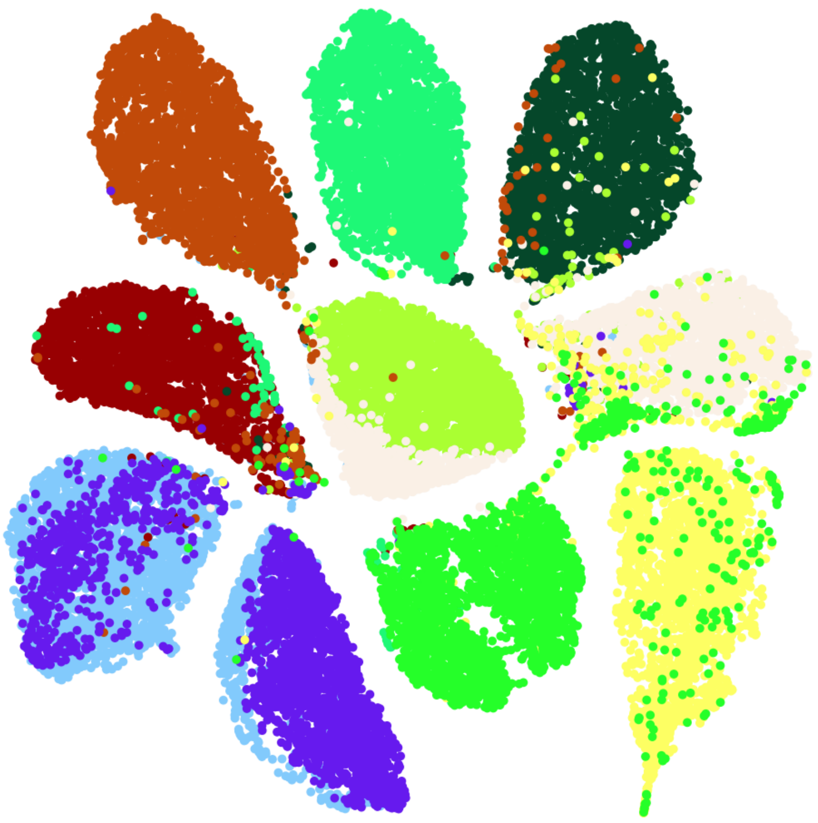

Figure 5.3: This figure illustrates the optimal clustering obtained by the DCSS algorithm on the EMNIST Balanced Letters (A-J) dataset.

5.5.4 EMNIST Balanced Letters (K-T)

Table 5.8: Statistical Analysis of the NMI on the EMNIST Balanced Letters (K-T) dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.51	0.00	0.51	0.51
AE + k-means	0.72	0.02	0.68	0.75
DCN	0.79	0.02	0.77	0.83
DEC	0.85	0.02	0.83	0.89
IDEC	0.84	0.03	0.77	0.87
DCSS	**0.88**	0.02	0.84	0.91

Table 5.9: Statistical Analysis of the ARI on the EMNIST Balanced Letters (K-T) dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.43	0.00	0.43	0.43
AE + k-means	0.66	0.03	0.61	0.70
DCN	0.72	0.03	0.68	0.80
DEC	0.82	0.02	0.80	0.88
IDEC	0.81	0.04	0.70	0.85
DCSS	**0.85**	0.03	0.78	0.89

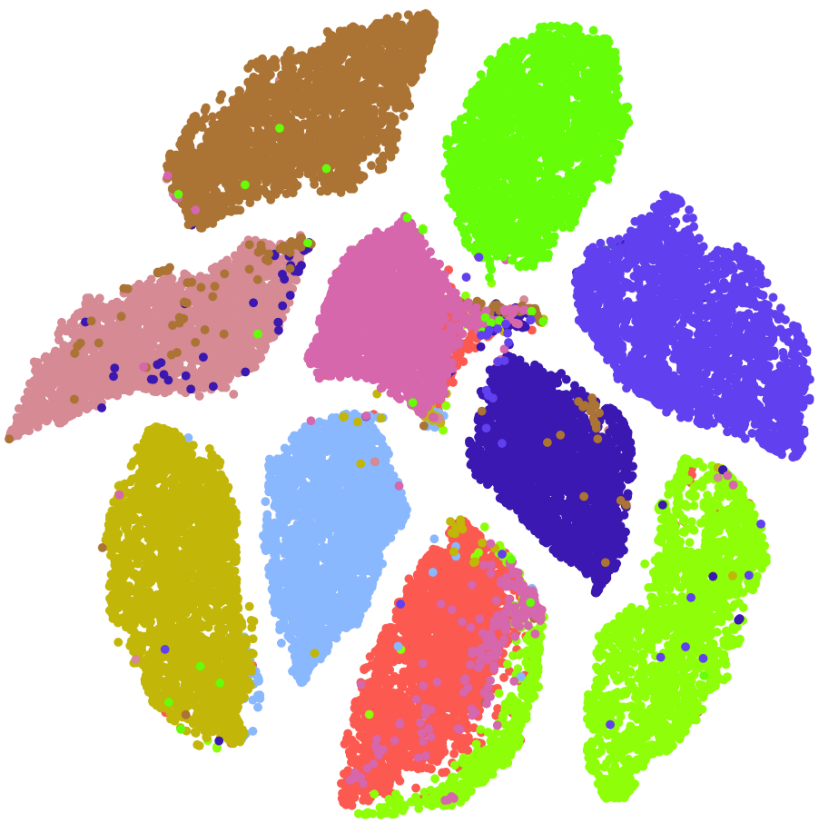

Figure 5.4: This figure illustrates the optimal clustering obtained by the DCSS algorithm on the EMNIST Balanced Letters (K-T) dataset.

5.5.5 EMNIST Balanced Letters (U-Z)

Table 5.10: Statistical Analysis of the NMI on the EMNIST Balanced Letters (U-Z) dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.47	0.00	0.47	0.47
AE + k-means	0.62	0.02	0.60	0.65
DCN	0.67	0.02	0.64	0.71
DEC	0.69	0.04	0.61	0.75
IDEC	0.71	0.04	0.65	0.76
DCSS	**0.74**	0.03	0.71	0.78

Table 5.11: Statistical Analysis of the ARI on the EMNIST Balanced Letters (U-Z) dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.41	0.00	0.41	0.41
AE + k-means	0.59	0.02	0.56	0.61
DCN	0.63	0.04	0.58	0.69
DEC	0.66	0.05	0.55	0.75
IDEC	0.67	0.05	0.62	0.76
DCSS	**0.71**	0.04	0.67	0.77

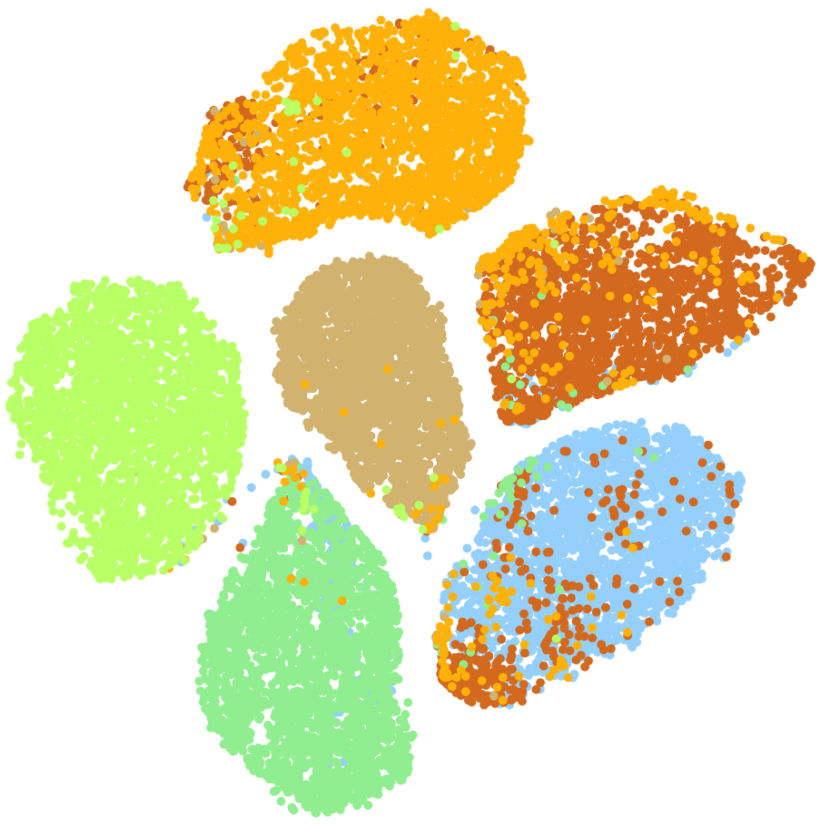

Figure 5.5: This figure illustrates the optimal clustering obtained by the DCSS algorithm on the EMNIST Balanced Letters (U-Z) dataset.

5.5.6 HAR

Table 5.12: Statistical Analysis of the NMI on the HAR dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.59	0.00	0.59	0.59
AE + k-means	0.71	0.03	0.62	0.73
DCN	0.74	0.04	0.63	0.77
DEC	0.67	0.08	0.55	0.79
IDEC	0.75	0.05	0.65	0.80
DCSS	**0.81**	0.05	0.69	0.85

Table 5.13: Statistical Analysis of the ARI on the HAR dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.46	0.00	0.46	0.46
AE + k-means	0.64	0.05	0.53	0.69
DCN	0.67	0.06	0.52	0.72
DEC	0.57	0.10	0.45	0.71
IDEC	0.68	0.06	0.53	0.75
DCSS	**0.73**	0.07	0.56	0.80

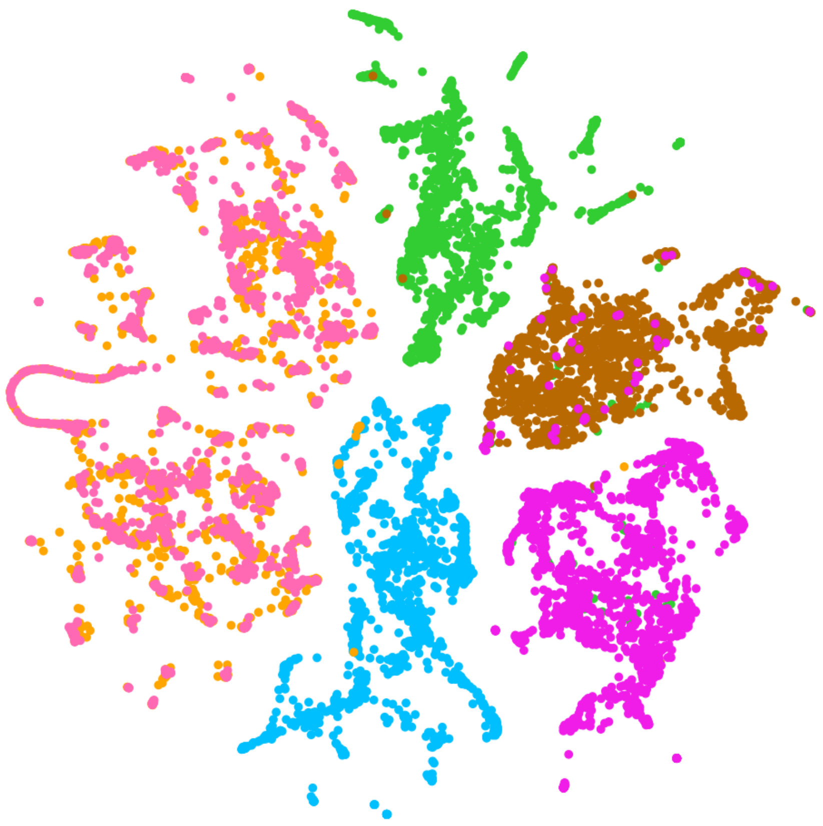

Figure 5.6: This figure illustrates the optimal clustering obtained by the DCSS algorithm on the HAR dataset.

5.5.7 Pendigits

Table 5.14: Statistical Analysis of the NMI on the Pendigits dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.69	0.01	0.68	0.69
AE + k-means	0.68	0.02	0.65	0.71
DCN	0.72	0.02	0.69	0.75
DEC	0.73	0.03	0.68	0.78
IDEC	**0.77**	0.02	0.74	0.81
DCSS	**0.77**	0.02	0.74	0.80

Table 5.15: Statistical Analysis of the ARI on the Pendigits dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.56	0.04	0.53	0.60
AE + k-means	0.57	0.04	0.51	0.64
DCN	0.62	0.04	0.54	0.67
DEC	0.61	0.05	0.55	0.71
IDEC	0.65	0.03	0.62	0.68
DCSS	**0.67**	0.03	0.62	0.71

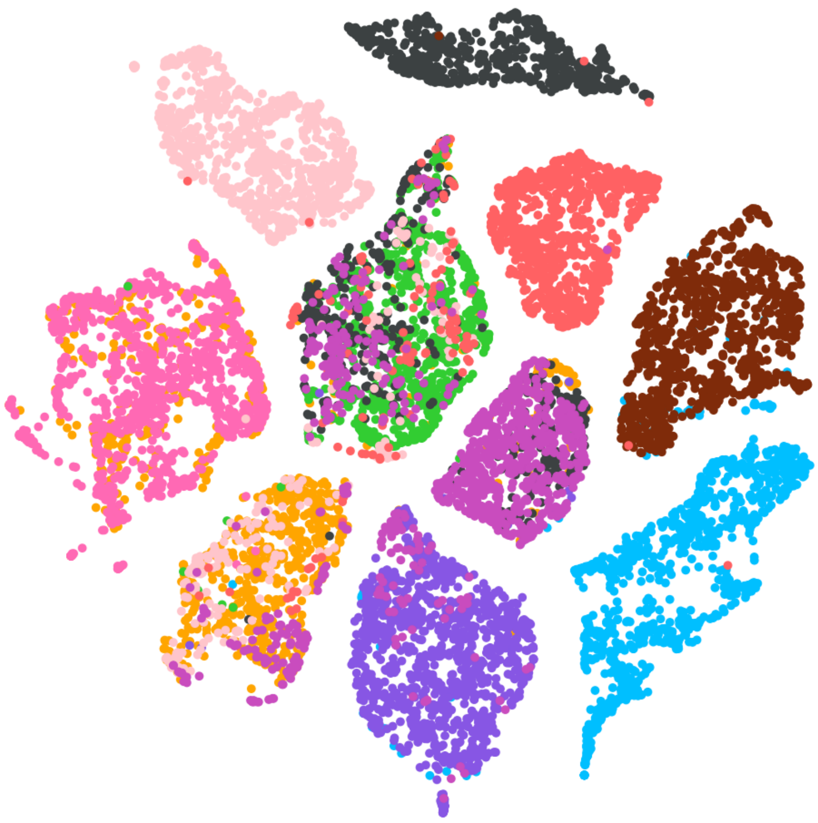

Figure 5.7: This figure illustrates the optimal clustering obtained by the DCSS algorithm on the Pendigits dataset.

5.5.8 Waveform-v1

Table 5.16: Statistical Analysis of the NMI on the Waveform-v1 dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.74	0.00	0.74	0.74
AE + k-means	0.86	0.10	0.65	0.95
DCN	0.87	0.11	0.67	1.00
DEC	0.86	0.11	0.65	0.99
IDEC	0.98	0.05	0.83	1.00
DCSS	**0.99**	0.01	0.97	1.00

Table 5.17: Statistical Analysis of the ARI on the Waveform-v1 dataset.

Algorithm	Mean	St. Dev.	Min	Max
k-means	0.70	0.00	0.70	0.70
AE + k-means	0.90	0.09	0.70	0.98
DCN	0.89	0.11	0.66	1.00
DEC	0.88	0.11	0.64	0.99
IDEC	0.98	0.06	0.81	1.00
DCSS	**1.00**	0.01	0.98	1.00

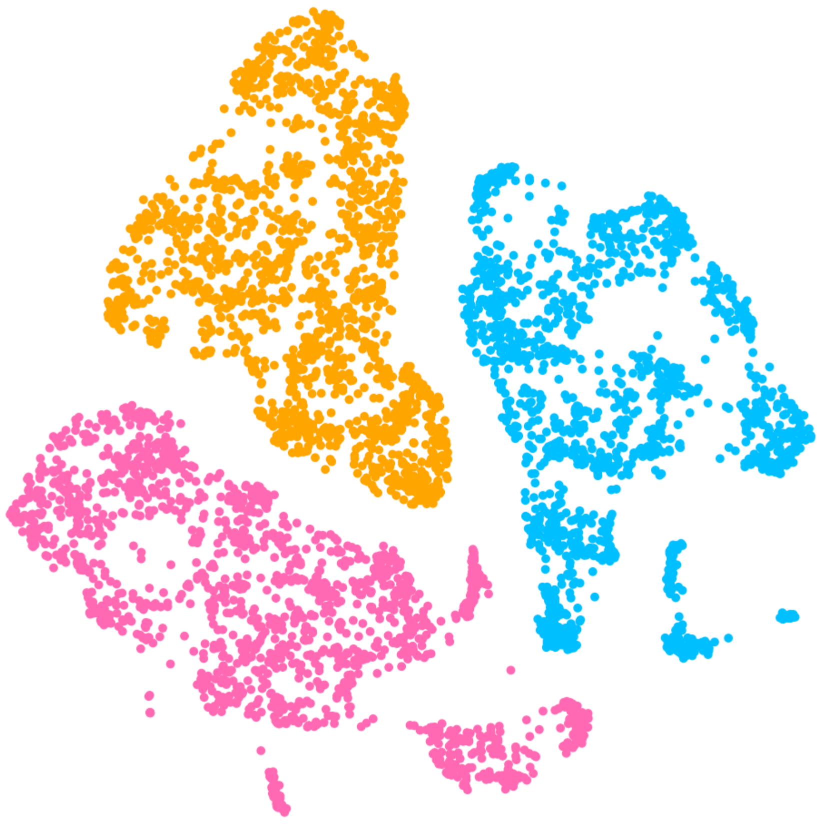

Figure 5.8: This figure illustrates the optimal clustering obtained by the DCSS algorithm on the Waveform-v1 dataset.

5.6 Analysis of Experimental Results

This section presents an analysis of the performance of the examined deep clustering methods across the datasets utilized in this study. As predicted, the application of

k-means clustering to a low-dimensional embedded space yielded enhanced performance, as evidenced by the outcomes of the AE+k-means approach in comparison to the conventional k-means methodology.

In general, DEC outperforms DCN on all datasets, with the exception of HAR and WVF-v1, where DCN demonstrates comparatively superior performance. It is notable that IDEC achieves superior results to DEC in the majority of cases, with some exhibiting a considerable advantage. This indicates that integrating the decoder into the clustering optimization process can lead to enhanced performance outcomes.

Moreover, the findings illustrate the efficacy of the presented DCSS method across all datasets. With regard to the NMI, the DCSS method demonstrates a notable enhancement, with values that range from 0.01 to 0.06 higher than those observed for other methods. Similarly, with regard to ARI, DCSS displays superior performance relative to other methods, exhibiting an improvement range of 0.01 to 0.05. Finally, these outcomes illustrate that soft silhouette represents a superior deep clustering function objective, as it enables the generation of cluster-friendly representations through the optimization of compact and well-separated clusters.

Chapter 6

Afterword

6.1 Conclusion

This thesis elaborates on the soft silhouette, an extension of the traditional silhouette score that incorporates probabilistic clustering assignments and studies a deep clustering methodology, DCSS [15], based on an AE architecture, to optimize this score. The DCSS approach directs the learned latent representations to form clusters that are both compact and well-separated. This dual focus on compactness and separability is essential in real-world applications, as it ensures that the resulting clusters are not only tightly grouped but also clearly distinct from one another.

The DCSS method has been subjected to rigorous testing and comparison against established deep clustering methods across a variety of benchmark datasets, yielding highly satisfactory results. The experimental findings indicate that the soft silhouette represents a more effective deep clustering objective function, significantly enhancing the quality of the learned representations within the embedded space, thereby improving clustering performance.

6.2 Suggestions for Future Work

There are several areas for future work, including the enhancement of clustering results through data augmentation techniques, which have been demonstrated to be effective for improving learned representations [81, 91]. Additionally, more sophisticated models and training methodologies, such as ensemble models [92] or adversarial learning [93], could be utilized. Another potential avenue for exploration is the modification of the learning procedure to incorporate self-paced learning [94], as prioritizing the learning of "easier" data is expected to yield better clustering outcomes [39, 95, 96]. However, our principal objective is to extend the DCSS algorithm to estimate the number of clusters by employing unimodality tests, in a manner analogous to the approaches utilized in the dip-means algorithm [97] and DIPDECK [98].

Bibliography

[1] H. Wang, C. Ma, and L. Zhou, "A brief review of machine learning and its application," *2009 International Conference on Information Engineering and Computer Science*, pp. 1–4, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:14271547

[2] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, pp. 264–323, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:12744045

[3] S. P. Lloyd, "Least squares quantization in pcm," *IEEE Trans. Inf. Theory*, vol. 28, pp. 129–136, 1982. [Online]. Available: https://api.semanticscholar.org/CorpusID:10833328

[4] M. E. Celebi, H. A. Kingravi, and P. A. Vela, "A comparative study of efficient initialization methods for the k-means clustering algorithm," *ArXiv*, vol. abs/1209.1960, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:6954668

[5] G. Vardakas and A. C. Likas, "Global k-means++: an effective relaxation of the global k-means clustering algorithm," *ArXiv*, vol. abs/2211.12271, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253761250

[6] B. Yang, X. Fu, N. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *International Conference on Machine Learning*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:6128905

[7] A. K. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques

and developments," *Neurocomputing*, vol. 267, pp. 664–681, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:31832898

[8] R. Suganya and R. M. Shanthi, "Fuzzy c- means algorithm- a review," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:17666771

[9] K. P. Murphy, "Machine learning - a probabilistic perspective," in *Adaptive computation and machine learning series*, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:17793133

[10] L. Vendramin, R. J. G. B. Campello, and E. R. Hruschka, "Relative clustering validity criteria: A comparative overview," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 3, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:17467072

[11] T. Szandała, "Review and comparison of commonly used activation functions for deep neural networks," *ArXiv*, vol. abs/2010.09458, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:224714035

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106

[13] K.-L. Du, "Clustering: A neural network approach," *Neural networks : the official journal of the International Neural Network Society*, vol. 23 1, pp. 89–107, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:15521864

[14] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987. [Online]. Available: https://api.semanticscholar.org/CorpusID:189900

[15] G. Vardakas, I. Papakostas, and A. Likas, "Deep clustering using the soft silhouette score: Towards compact and well-separated clusters," *ArXiv*, vol. abs/2402.00608, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267365548

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:195908774

[17] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, “Contractive auto-encoders: Explicit invariance during feature extraction,” in *International Conference on Machine Learning*, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:8141422

[18] G. C. Nutakki, B. Abdollahi, W. Sun, and O. Nasraoui, “An introduction to deep clustering,” *Clustering Methods for Big Data Analytics*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:69919246

[19] 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. [Online]. Available: https://api.semanticscholar.org/CorpusID:206491647

[20] E. Aljalbout, V. Golkov, Y. Siddiqui, and D. Cremers, “Clustering with deep learning: Taxonomy and new methods,” *ArXiv*, vol. abs/1801.07648, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:8400105

[21] S. Haykin, *Neural Networks and Learning Machines*, ser. Neural networks and learning machines. Prentice Hall, 2009, no. v. 10. [Online]. Available: https://books.google.gr/books?id=K7P36lKzI_QC

[22] GeeksforGeeks, “Autoencoders,” https://www.geeksforgeeks.org/auto-encoders/, accessed: April 10, 2024.

[23] H. Lee, R. B. Grosse, R. Ranganath, and A. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *International Conference on Machine Learning*, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:12008458

[24] G. E. Hinton, “A practical guide to training restricted boltzmann machines,” in *Neural Networks*, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:21145246

[25] A. Sharif Ahmadian, "Chapter 7 - numerical modeling and simulation," in *Numerical Models for Submerged Breakwaters*, A. Sharif Ahmadian, Ed. Boston: Butterworth-Heinemann, 2016, pp. 109–126. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128024133000079

[26] H. Faris, I. Aljarah, and S. Mirjalili, "Chapter 28 - evolving radial basis function networks using moth–flame optimizer," in *Handbook of Neural Computation*, P. Samui, S. Sekhar, and V. E. Balas, Eds. Academic Press, 2017, pp. 537–550. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128113189000284

[27] S. Saito and R. T. Tan, "Neural clustering: Concatenating layers for better projections," 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:108389883

[28] K. Li and J. Malik, "Implicit maximum likelihood estimation," *ArXiv*, vol. abs/1809.09087, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52816401

[29] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *Neural Information Processing Systems*, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:261560300

[30] 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

[31] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International Conference on Machine Learning*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:6779105

[32] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," 07 2000. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/347090.347176

[33] 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, pp. 421–429, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:21780790

[34] G. Chen, "Deep learning with nonparametric clustering," *ArXiv*, vol. abs/1501.03084, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:18134251

[35] P. Huang, Y. Huang, W. Wang, and L. Wang, "Deep embedding network for clustering," *2014 22nd International Conference on Pattern Recognition*, pp. 1532–1537, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:18509954

[36] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5147–5156, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:8105340

[37] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, "Clustergan : Latent space clustering in generative adversarial networks," in *AAAI Conference on Artificial Intelligence*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52188737

[38] A. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Neural Information Processing Systems*, 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:18764978

[39] F. Li, H. Qiao, B. Zhang, and X. Xi, "Discriminatively boosted image clustering with fully convolutional auto-encoders," *Pattern Recognit.*, vol. 83, pp. 161–173, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:14958831

[40] Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1798–1828, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:393948

[41] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[42] S. Zhou, H. Xu, Z. Zheng, J. Chen, Z. li, J. Bu, J. Wu, X. Wang, W. Zhu, and M. Ester, “A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions,” *ArXiv*, vol. abs/2206.07579, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249674426

[43] Y. Ren, J. Pu, Z. Yang, J. Xu, G. Li, X. Pu, P. S. Yu, and L. He, “Deep clustering: A comprehensive survey,” *ArXiv*, vol. abs/2210.04142, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252780393

[44] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *CoRR*, vol. abs/1312.6114, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:216078090

[45] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, “Graph neural networks: A review of methods and applications,” *ArXiv*, vol. abs/1812.08434, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:56517517

[46] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, “Variational deep embedding: An unsupervised and generative approach to clustering,” in *International Joint Conference on Artificial Intelligence*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:2546662

[47] G. Vardakas and A. C. Likas, “Implicit maximum likelihood clustering,” in *Artificial Intelligence Applications and Innovations*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249872450

[48] ——, “Neural clustering based on implicit maximum likelihood,” *Neural Computing and Applications*, vol. 35, pp. 21 511–21 524, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258045111

[49] W. Xu, X. Liu, and Y. Gong, “Document clustering based on non-negative matrix factorization,” *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:2237682

[50] D. Cai, X. He, and J. Han, “Locally consistent concept factorization for document clustering,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 902–913, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:10512564

[51] G. de Soete and J. D. Carroll, "K-means clustering in a low-dimensional euclidean space," 1994. [Online]. Available: https://api.semanticscholar.org/CorpusID:124096286

[52] V. M. Patel, H. V. Nguyen, and R. Vidal, "Latent space sparse subspace clustering," *2013 IEEE International Conference on Computer Vision*, pp. 225–232, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:7094253

[53] B. Yang, X. Fu, and N. Sidiropoulos, "Learning from hidden traits: Joint factor analysis and latent clustering," *IEEE Transactions on Signal Processing*, vol. 65, pp. 256–269, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:5104944

[54] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt, and T. Hoffman, Eds., vol. 19. MIT Press, 2006. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2006/file/5da713a690c067105aeb2fae32403405-Paper.pdf

[55] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:205001834

[56] D. Sculley, "Web-scale k-means clustering," in *The Web Conference*, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:6634147

[57] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv*, vol. abs/1502.03167, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:5808102

[58] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:215827080

[59] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *ArXiv*, vol. abs/1311.2901, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:3960646

[60] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:1629541

[61] F. D. la Torre and T. Kanade, "Discriminative cluster analysis," *Proceedings of the 23rd international conference on Machine learning*, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:5296934

[62] J. Ye, Z. Zhao, and M. Wu, "Discriminative k-means for clustering," in *Neural Information Processing Systems*, 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:2366213

[63] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, pp. 395–416, 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:3264198

[64] Y. Yang, D. Xu, F. Nie, S. Yan, and Y. Zhuang, "Image clustering using local discriminant models and global integration," *IEEE Transactions on Image Processing*, vol. 19, pp. 2761–2773, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:4523132

[65] F. Nie, Z. Zeng, I. W.-H. Tsang, D. Xu, and C. Zhang, "Spectral embedded clustering: A framework for in-sample and out-of-sample spectral clustering," *IEEE Transactions on Neural Networks*, vol. 22, pp. 1796–1808, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:8297836

[66] L. van der Maaten and G. E. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:5855042

[67] L. van der Maaten, "Learning a parametric embedding by preserving local structure," in *International Conference on Artificial Intelligence and Statistics*, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18203584

[68] M. L. Freimer and R. Bellman, "Adaptive control processes: A guided tour," *The Mathematical Gazette*, vol. 46, pp. 160 – 161, 1961. [Online]. Available: https://api.semanticscholar.org/CorpusID:64832941

[69] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991. [Online]. Available: https://api.semanticscholar.org/CorpusID:7343126

[70] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:17804904

[71] K. Nigam and R. Ghani, "Analyzing the effectiveness and applicability of co-training," in *International Conference on Information and Knowledge Management*, 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:7464925

[72] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647

[73] Q. V. Le, M. Ranzato, R. Monga, M. Devin, G. S. Corrado, K. Chen, J. Dean, and A. Ng, "Building high-level features using large scale unsupervised learning," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8595–8598, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:206741597

[74] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6844431

[75] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation," in *International Joint Conference on Artificial Intelligence*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:39311659

[76] X. Peng, S. Xiao, J. Feng, W.-Y. Yau, and Z. Yi, "Deep subspace clustering with sparsity prior," in *International Joint Conference on Artificial Intelligence*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:17770085

[77] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[78] Y. LeCun and C. Cortes, "The mnist database of handwritten digits," 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:60282629

[79] G. Cohen, S. Afshar, J. C. Tapson, and A. van Schaik, "Emnist: Extending mnist to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:30587588

[80] A. U. Asuncion, "Uci machine learning repository, university of california, irvine, school of information and computer sciences," 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:203706180

[81] X. Guo, E. Zhu, X. Liu, and J. Yin, "Deep embedded clustering with data augmentation," in *Asian Conference on Machine Learning*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:53415432

[82] Y. Ren, N. Wang, M. Li, and Z. Xu, "Deep density-based image clustering," *Knowl. Based Syst.*, vol. 197, p. 105841, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:54486714

[83] W. Guo, K. Lin, and W. Ye, "Deep embedded k-means clustering," *2021 International Conference on Data Mining Workshops (ICDMW)*, pp. 686–694, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:238227137

[84] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:15539264

[85] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:13740328

[86] E. Rendón, I. Abundez, A. Arizmendi, and E. M. Quiroz, "Internal versus external cluster validation indexes," 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:16604539

[87] P. A. Estévez, M. Tesmer, C. A. Pérez, and J. M. Zurada, "Normalized mutual information feature selection," *IEEE Transactions on Neural Networks*, vol. 20, pp.

189–201, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:6340275

[88] L. J. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, pp. 193–218, 1985. [Online]. Available: https://api.semanticscholar.org/CorpusID:189915041

[89] J. E. Chac'on and A. I. Rastrojo, "Minimum adjusted rand index for two clusterings of a given size," *Advances in Data Analysis and Classification*, vol. 17, pp. 125–133, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:211069524

[90] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971. [Online]. Available: http://www.jstor.org/stable/2284239

[91] X.-X. Deng, D. Huang, D. Chen, C. Wang, and J. Lai, "Strongly augmented contrastive clustering," *Pattern Recognit.*, vol. 139, p. 109470, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249240508

[92] S. Affeldt, L. Labiod, and M. Nadif, "Spectral clustering via ensemble deep autoencoder learning (sc-edae)," *Pattern Recognit.*, vol. 108, p. 107522, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:57721329

[93] X. Yang, C. Deng, K.-J. Wei, J. Yan, and W. Liu, "Adversarial learning for robust deep clustering," in *Neural Information Processing Systems*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:227275327

[94] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Neural Information Processing Systems*, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:1977996

[95] X. Guo, X. Liu, E. Zhu, X. Zhu, M. Li, X. Xu, and J. Yin, "Adaptive self-paced deep clustering with data augmentation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, pp. 1680–1693, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:145883156

[96] K. Zhang, C. Song, and L. Qiu, "Self-paced deep clustering with learning loss," *Pattern Recognit. Lett.*, vol. 171, pp. 8–14, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258504613

[97] A. Kalogeratos and A. C. Likas, "Dip-means: an incremental clustering method for estimating the number of clusters," in *Neural Information Processing Systems*, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:5695121

[98] C. Leiber, L. G. M. Bauer, B. Schelling, C. Böhm, and C. Plant, "Dip-based deep embedded clustering with k-estimation," *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236980202

Short Biography

Ioannis Papakostas was born in Ioannina, Greece, in 1999. He commenced his academic career in 2017 by enrolling in the undergraduate program of Computer Science and Engineering at the University of Ioannina, where he obtained his diploma in 2022. His diploma thesis, entitled ”Deep Clustering with the Soft Silhouette Index,” demonstrated his early interest in machine learning, particularly in the domain of clustering. He subsequently enrolled in the postgraduate program of the same department in 2023. His research interests are primarily focused on machine learning, with a specific emphasis on clustering methods.