Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

Unsupervised clustering of MDS data using federated learning

Supervisor:
Prof. Enrico Giampieri

Co-supervisor:
Prof. Gastone Castellani

Submitted by:
Lorenzo Sani

Academic Year 2021/2022

Abstract

In this master thesis we developed a model for unsupervised clustering on a data set of biomedical data. This data has been collected by GenoMed4All consortium from patients affected by Myelodysplastic Syndrome (MDS), that is an haematological disease. The main focus is put on the genetic mutations collected that are used as features of the patients in order to cluster them. Clustering approaches have been used in several studies concerning haematological diseases such MDS. A neural network-based model was used to solve the task. The results of the clustering have been compared with labels from a "gold standard" technique, i.e. hierarchical Dirichlet processes (HDP). Our model was designed to be also implemented in the context of federated learning (FL). This innovative technique is able to achieve machine learning objective without the necessity of collecting all the data in one single center, allowing strict privacy policies to be respected. Federated learning was used because of its properties, and because of the sensitivity of data. Several recent studies regarding clinical problems addressed with machine learning endorse the development of federated learning settings in such context, because its privacy preserving properties could represent a cornerstone for applying machine learning techniques to medical data. In this work will be then discussed the clustering performance of the model, and also its generative capabilities.

Contents

Chapter 1

Introduction

The aim of this work is to solve the clustering problem of a specific data set, namely EUROMDS, of biomedical data exploiting an emerging decentralized technique of Machine Learning (ML), i.e. Federated Learning (FL). The motivation for using this novel technique is that, as I will deeply explain below, it allows the participation of multiple centers, both at training and inference steps, without having to actually move or see the data, and then protecting from privacy risks. Moreover FL has shown to be promising in reaching a ML objective in a decentralized setting without having to touch or see data, and respects privacy policies introduced for these, e.g. by General Data Protection Regulation (GDPR) [1] in European Union. FL can be used to resolve privacy issues and mitigate the risk of a data breach in clinical information, since transfer and centralization of data are not required. Privacy protection is particularly beneficial for medical data analysis, since medical data represents one of the most sensitive types of personal data. To protect patients' privacy, deidentification methods have typically been applied [2, 3, 4]. However, data centralization is required for both deidentifying data and evaluating the risk of reidentification. If the data is centralized, the risk of a data breach is increased. Moreover, when deidentifying the data set, the direct or indirect identifiers in the medical data must be determined. This is challenging because of the lack of clear guidelines. The Health Insurance Portability and Accountability Act (HIPAA) in the United States provides clear deidentification guidance; it defines 18 types of protected health information to be removed [5]. However, many researchers and social activists claim that this guidance should be revised to enhance privacy protection [6]. In contrast, FL does not require the centralization of raw data. As a result, even the FL developers cannot access the raw data. Therefore, FL can solve privacy or deidentification issues that occur when using clinical data. We have developed our federated learning context in a "simulated" mode, where data is being distributed on nodes that cannot see other nodes' data. We say "simulated" because we had actually the possibility to see and touch data, which would not be allowed in a truly federated learning setting. However our implementation have been developed to be exploited in an actual federated setting.

Interest and researches in FL have grown exponentially since it was first introduced, and, because of its privacy protection features, it represents a solid breakthrough for medical application of ML. The task chosen for this application, unsupervised clustering, is motivated by the fact that it is widely used in medical applications for patient stratification. It is, in fact, often a first step in many medical procedure for diagnosis and prognosis. The choice of the principal data set for study was motivated by the fact that in myeloid malignancies classification is an important task, that requires collecting data from numerous centers. The principal classifications are on the basis of clinical and morphological criteria, but these are often successfully complemented by introducing genomic features as these are closer to the disease biology and better capture clinical pathological entities.

1.1 Clustering

Clustering analysis or, commonly, clustering is the task of grouping a set of objects in such a way that objects in the same group, i.e. a cluster, are more similar to each other than those in other groups. It is a main task of exploratory data analysis and a common technique for statistical data analysis, also used in many fields, including pattern recognition, image analysis, information retrieval, bioinformatics, data compression, computer graphics and machine learning. This could be achieved by various algorithms that could differ significantly in identifying what constitutes a cluster and how to effectively find them. Notions of cluster include groups with small distances between members, dense areas of data space, intervals or particular statistical distributions. Many of these notions come from an effective definition of distance in data space, density or limiting values. Therefore clustering can be formulated as a multi-objectives optimization problem, in fact it usually involves the definition of multiple criteria that drive the decision of whether to cluster or not data points. Every algorithm or model for clustering comes with a different definition of cluster, then the appropriate clustering algorithm and parameter setting to use depends on the data set and on the intended use of the result. Cluster analysis in not an automatic task, but a process of knowledge discovery that iteratively tries to optimize the multi-objectives involving trial and failures. It is often necessary to preprocess data and model parameters in order to achieve a result with the desired properties.

1.1.1 Popular clustering Methods

Since the notion of "cluster" cannot be precisely defined, different algorithms or models try to learn the clustering of data based on different definition with a common denominator: group of data objects. The differences between these methods are reflected in the properties of the final results. Clustering algorithms can be classified into the following

categories, according to the method used in [7]:

Hierarchical clustering methods: these proceed successively by either merging smaller clusters into larger ones, or by splitting larger clusters and produce the so called dendrogram. They are also classified as "connectivity-based" clustering methods. Popular choices are single-linkage clustering, complete linkage clustering, and UPGMA or WPGMA ("Unweighted or Weighted Pair Group Method with Arithmetic Mean") [8]

Partitioning methods: these methods attempt to directly decompose the data set into a set of disjoint clusters. These are preferable to hierarchical methods in applications involving large data sets for which the construction of the dendrogram is computationally prohibitive. Density-based methods are a subcategory of these, they try to group neighbouring objects into clusters based on density conditions. Methods of this category depend strongly on the concept of distance and on the optimization of a cost function measuring the goodness of the decomposition.

Mixture resolving methods: these are based on the assumption that the patterns to be clustered are drawn from one of several distributions, the goal is to identify the parameters of each and (perhaps) their number. Mixture Gaussian models are the most popular.

Nearest neighbours methods: these are based on the notion of nearest neighbour distance, and share many properties with partitioning methods. They usually assign labels to data iterating a threshold checking on the nearest neighbour distance. The famous K-means method belongs to this category.

Fuzzy clustering: the main characteristic of these methods is that they allow for the same object to be included into different clusters with various degree of memberships. They are then strongly dependent on the membership function adopted.

Artificial neural networks methods: these methods are based on neural network models that try to identify clusters by optimizing a properly defined loss function. Competitive neural networks, e.g. GANs [9], or deep autoencoders coupled with other methods, e.g. DEC [10], are used to solve this task.

Evolutionary methods: these, inspired by natural evolution, make use of evolutionary operators and a population of solutions to obtain the globally optimal partition of the data. The main operators are: selection, recombination, and mutation. The results are evaluated by a fitness function.

Search based methods: these techniques used to obtain the optimal value of the criterion function using deterministic or stochastic approaches.

Moreover, clustering methods can be also classified on the basis of their membership function. Whenever the method is able to assign a class label exclusively to each data point we are dealing with"hard clustering", instead when a data point could be assigned to more than one cluster (perhaps with probability) we talk about "soft clustering". There exist many methods than combine the notions explained above.

A core aspect of clustering is dimensionality reduction, a key concept in machine learning and data science. Dimensionality reduction is important for feature extraction and data visualization, especially in the case of high-dimensional data. Among various methods, we can mention those based on nonlinear projection of high-dimensional data into a lower dimensional space, such as distributed stochastic neighbor embedding (t-SNE) [11], and those based on manifold learning, such as uniform manifold approximation and projection for dimension reduction (UMAP) [12].

To our knowledge, both of these algorithms have not yet been implemented in a federated way, but they can constitute a valuable benchmark. Because of the limitations on the use of data in a federated setting, solving a clustering task could become a tricky task, if not unfeasible, using the standard methods. Since it was first developed, federated learning has shown promising results to solve machine learning objective consisting in the optimization of neural networks (NN). Therefore, NN models with a clustering objective are the most suitable in a federated setting.

1.1.2 Clustering MDS

Clustering techniques in medical context are extensively used for patient stratification. In general, patients can be stratified according to some features obtained by multi-omics measurements (genomic, imaging, etc.). After this step, we try to predict some clinical outcome. This approach was applied in several studies concerning haematological diseases [13]. In [14] is described a specific method for the Myelodysplastic syndrome (MDS). In this report, the authors used a particular type of clustering: the hierarchical Dirichlet processes (HDP), a non parametric Bayesian approach to clustering grouped data. Due to the high number of studies that used this method, and also on the basis of the numerous clinical validations, the HDP method has become a sort of "gold standard" for clustering of hematological malignancies. It has to be noted that the choice of HDP is motivated also by the nature of the data set. As I will specify later, the data set format is, roughly speaking, formed by (for each patient) a set of genomic measurements (obtained from a gene panel), a set of cytogenetic measurements and by a third set of clinical variables. For the sake of simplicity, we will denote the genomic and cytogenetic measurements with the common term of "mutations". According to this simplification, the first two sets of measurements can be coded by using a 0 (absence of mutation) or a 1 (presence of mutation). In this way, each row (each patient) will be represented as a realization of a multinomial distribution (the conjugate prior of the Dirichlet distribution). Another advantage of the HDP is that the number of clusters is determined

automatically and there is no need of an assignment a priori. Unfortunately, to our current knowledge, there is no federated implementation for such algorithm, also a "ex novo" federated implementation is not easy to develop. The current state of the art in the federated implementation for similar algorithms is the realization of a federated framework for the so called latent Dirichlet allocation (LDA) [15], a generative statistical model that has also been used for applications in hematological malignancies. We observe that the HDP is the non-parametric Bayesian "natural extension" of the LDA, and that the number of clusters can be learnt from the data. For these reasons, in order to have a clinical interpretation and translation, all the federated implementations have to be compared not only with a centralized implementation, but also with the HDP results.

It is now important to highlight the fact that a federated implementation of a machine learning clustering method could represent a cornerstone in the medical context, in which data involved is usually highly sensitive and it is very difficult to share because of privacy restriction. Federated learning methods allow to train on data without actually "seeing" it, meaning that these algorithms do not need a complex sharing-data procedure, since data could be kept in the same place it was collected (e.g. clinics, hospitals, even mobile devices). Initial research results show that the performance of a federated learning algorithm is comparable to its correspondent in a data-sharing context between medical institutions, at the same time it is much less prone to privacy concerns [16]. Many challenges yet exist to guarantee a completely secure procedure, on the other hand different approaches have been proposed to tackle some of the issues [17].

1.2 Federated Learning

Federated Learning is a machine learning setting where many clients, or nodes, collaboratively train a model under the orchestration of a central server, or a service provider, while keeping the training data decentralized [18]. Such clients that participate the training are usually represented by edge-devices, as smartphones or wearables, or by whole organizations, as hospitals or consortia. FL embodies the principles of focused collection and data minimization, and can mitigate many of the systemic privacy risks and costs resulting from traditional, centralized machine learning. The term federated learning was introduced in 2016 [19]: "We term our approach Federated Learning, since the learning task is solved by a loose federation of participating devices (which we refer to as clients) which are coordinated by a central server." An unbalanced and non-IID (identically and independently distributed) data partitioning across a massive number of unreliable devices with limited communication bandwidth was introduced as the defining set of challenges. New challenges related to FL have arisen in the last few years of intense research, e.g. convergence of the optimization algorithms, improved security regarding data privacy protection, protection against adversial clients or servers, defence against attacks or failures, guarantee of fairness and effectiveness, provision for local

personalization, addressing system challenges.

1.2.1 Main differences w.r.t. distributed learning

Across the whole range of machine learning methods, there existed some for distributing the learning across clients or nodes before developing federated learning. Those methods, usually classified as "distributed learning" possess some differences w.r.t. federated learning that are important to highlight. Usually distributed learning applies to a "flat" centralized data set that is being distributed among computation nodes for in order to obtain a more efficient training. Any client can thus read any part of the data set, in fact algorithms for distributed learning optimize the distribution and the balance of samples for efficiency. The computation is more often the bottleneck in these situations, while the communication often happens between different nodes in the same data centers. Clients are also stateful, meaning that they keep the state of the training at every iteration, and can be directly addressed to by the orchestrator.

1.2.2 Cross-device federated learning

FL was initially deployed on mobile or edge-devices in a context formally known as "cross-device"". The characteristics of such context are peculiar. Only a fraction of clients is available at any given time, often with diurnal or other variations. The setting is massively parallel, in fact large scale applications have reached up to 10^{10} clients. Communication is often the primary bottleneck, although it depends on the task, since wi-fi or slower connections are used. Clients here cannot be indexed, i.e. no use of any client identifiers, because they are too many, and they are also stateless, meaning that a fresh sample of never-before-seen clients at each round of computation is assumed. The state of a federated model is usually represented by the parameters of the model and/or the state of the optimizer. Clients are highly unreliable, more than 5% of those participating in a round of computation are expected to fail or drop out. Data partitioning is fixed and it is usually horizontal, i.e. clients provide different examples of data with the same shared features. These properties are mainly influenced by the device type used in such context, in fact mobile or edge-device are usually selected for participating in a computation round only when they are idle, connected to the network and have a high level of battery.

1.2.3 Cross-silo federated learning

While some "cross-device" medical applications have been proposed [20], the majority of biomedical data are collected in hospitals or clinics, not on mobile or edge-devices. The FL context in which training is performed on siloed data is formally known as "cross-silo" setting. Clients here are usually organizations, e.g. medical or financial, or

geo-distributed data centers. Many applications have been proposed in this context, e.g. [21, 22, 23, 24]. In these set situations, the local training of the model might be sufficient, but FL allows to increase the amount of information as many more samples can be used. This produces more effective and reliable models. Typically these settings have tens or hundreds of clients, differently from "cross-device". The primary bottleneck could be represented not only by the communication, that is in general more reliable and fast, e.g. fiber optics connections between network-optimized data centers, but also by computation, since there is much more data involved. Each client has an identity or name that allows the system to address it specifically. Clients are also stateful, i.e. each client should be able to participate in every round of computation while keeping at any time the state of the training. This setting produces relatively few failures since clients are more reliable. Data partition is fixed, and can be either horizontal or vertical, i.e. clients provide different features from the same example.

1.2.4 Life cycle of federated learning model

The FL process is usually performed by many actors. Adding to the already mentioned clients and server, there are an administrator, e.g. a researcher, a model engineer or a data analyst, and also the external world, i.e. other edge-devices, customers to which the final trained model could be deployed. Most of these customers are represented by the clients which participate the training, but there might be many more. The typical top level workflow of a FL process is:

Task identification. The administrator identifies the task to be solved with FL.

Client set up. When needed, the clients are provided with the resources necessary to store locally their training data and metadata.

Training simulation. The administrator may simulate using a proxy data set the FL training procedure for hyperparameters tuning and model architecture optimization.

Actual federated model training. Usually multiple tasks of federated training are started (in parallel) to train different flavors of the model, i.e. using different training procedures, hyperparameters.

Model evaluation. Once that the models have trained sufficiently (some metrics can ensure this), they are analyzed to select good candidates. Analysis of these models may include metrics computed on proxy data sets, or also federated evaluation wherein the models are pushed to held-out clients for evaluation on local client data.

Final deployment. Once a good model has been selected, it undergoes the usual launch process. This process is independent to the previous steps and it is set by the owner of the application. In practice, this step is equal to those applied in traditional centralized approaches.

1.2.5 Typical federated learning training process

It is now important to focus on the training process that is the most different step of the life cycle w.r.t. traditional centralized ML. FL research has developed many training algorithms that differ in some of the general procedures involved. Then, the general procedure will be described in the following, and some focused insights will be discussed.

A server, or a service provider, orchestrates the entire training process, through the iterative repetition of the following steps until the training is stopped. The stopping criteria are define at the discretion of the administrator that is monitoring the training process.

1. **Client selection:** the server samples from a set of available clients, whose availability is conditioned by some requirements. These requirements restrict heavily the eligibility of clients, especially in "cross-device" context, e.g. they may rely on the wi-fi connection quality, the idle state of the device, or whether the device is plugged in for charging. In "cross-silo", however, all the clients may be available at every iteration, or, alternatively, the administrator may have set for the server to wait until all the clients are available.

2. **Broadcast:** the selected clients download the current state of the FL training. This state consists at least of the model's parameters, but some training procedure could also include the optimizer's state or some state-dependent hyperparameters.

3. **Client computation:** each selected client locally computes a model update executing locally the training program. This local training has been set up by the administrator, and it may depend on the federated algorithm chosen, on the model itself, or on the state of the training. At this stage, one local model for every client is produced, while the global model is kept only at server place.

4. **Aggregation:** the server collects an aggregate of the device updates. The aggregation algorithm is crucial in order to reach the convergence of the training procedure. Many aggregation algorithms have been proposed in the last few years. These take into account efficiency optimization, secure aggregation for increased privacy, compression mechanisms, noise addition and update clipping for differential privacy.

5. **Model update:** the server locally updates the global model based on the aggregated update computed from the clients that effectively participated in the current round.

The "client selection" step is usually very different between "cross-device" and "cross-silo" approaches. In the first case, the algorithm must choose from a very high number of clients those for which the many conditions of reliability are satisfied. Usually it is important that the device is idle, connected to internet, connected to the charger, but many other conditions could be pinned causing the number of available clients to be much lower than the total in the setup. On the other hand "cross-silo" approaches may not limit in any way the number of clients participating the training as they are often reliable in terms of network availability and computational capabilities. The nodes in "cross-silo" are often represented by specialized machines that are devoted to perform the operations involved in a federated training.

"Client computation" step deserves a further brief discussion. The federated setup, in fact, must take into account the computational capabilities of the nodes involved during training. Thus developing the client computation step for "cross-device" federated learning might involve simpler and faster calculations than traditional centralized machine learning because of the computational resources edge-devices usually have. On the other hand, the computational capabilities of nodes involved in "cross-silo" federated learning are much greater than "cross-device", allowing researchers to exploit heavier pipelines and using the same tools of traditional centralized machine learning.

The aggregation procedure is the true core of federated learning. Despite absent mathematical guarantees of the convergence of such algorithms in nonconvex functions minimization, those involved in training neural network models, many proposed algorithms have been successful in applications. The most popular is Federated Averaging (FedAvg) algorithm, an adaptation of local-update or parallel Stochastic Gradient Descent (SGD). Here, each client runs a predetermined number of SGD steps locally, and then sends back to the server the parameters of its local model. Local models from clients' updates are then averaged by the server to form the updated global model. The pseudo-code in Algorithm 1 summarizes the procedure involving K clients making E local epochs whose batches size is fixed to B. There exist also other algorithms. Other algorithms used in this work are FedAdam and FedYogi [25] which will be discussed later along with their centralized counterpart.

Algorithm 1: *FedAvg.* The K clients are indexed by k; E is the number of local epochs; η is the learning rate; B is the size of the local batches; w indicates model parameters, when naked those of global mode, with superscript k those of k^{th} client, with subscript 0 those from the common initialization, subscripts t and $t+1$ indicate the current and the next iteration respectively.

Server executes:
- initialize w_0
- **for** *each federated round* $t = 1, 2, ...$ **do**
  - $S_t \leftarrow$ (select available clients)
  - **for** *each client* k *in* S_t **do in parallel**
    - $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  - **end**
  - $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
- **end**

ClientUpdate(k, w_t):
- $\mathcal{B} \leftarrow$(split k-th client's data set into batches of size B)
- **for** *each local epoch* i *from* 1 *to* E **do**
  - **for** *batch* $b \in \mathcal{B}$ **do**
    - $w \leftarrow w - \eta \nabla l(w; b)$
  - **end**
- **end**
- return w to server

1.2.6 Federated learning framework

In order to perform experiments or to actually implement a federated learning setup, it is necessary to use a framework. Federated learning frameworks are composed by a set of libraries that allows to customize the algorithmic part of the setup, while under the hood generalities about the training process are shared. Frameworks usually provide a simple and general pipeline that follows the main steps of a federated learning process. We then chose Flower Framework [26], because it provides the necessary coding structures for performing both simulation and real implementation. It is also scalable up to millions of clients maintaining a top trending performance. We were thus able to perform a simulation of a federated setting on a single node using "virtual clients", and real federated setting exploiting different nodes in different places.

Chapter 2

Methods

In this section, the methods used to assess the main task of the thesis work will presented and discussed. First the motivation for the choice of the method will be presented and justified. Then, a complete explanation of chosen the model will be accompanied by our contributions. These contributions will be explained along with the motivation that led us to introduce them.

2.1 Federated Clustering

Clustering plays a central role in disease outcome prediction and intervention planning by bridging the gap between traditional, average based medicine, and a fully personalized one. Clustering patients through using relevant attributes, e.g. clinical, phenotypical, demographical and omics, allows us to identify responses to therapy that might have been overlooked as unpredictable variability before. To make full use of this ability it is necessary to employ extended clinical studies, with potentially tens of thousands of patients involved. These kinds of studies are unfeasible for individual centers and, even when possible, cannot be generalized to a broader population (for example at the European level) due to biases in socio-demographical and ethnicity characteristics of the population under study. This means that multi-center studies are indispensable, but these kinds of studies have issues with privacy and deindentification of patients, as some of the information cannot be shared due to privacy concerns. These issues are even more severe if the centers are distributed across different European countries or, even worse, outside of the EU.

To circumvent these issues, federated methods can be applied to identify cross-regional clusters without private information leaking between the centers. Sadly, most clustering methods cannot be directly extended in a federated setting, and even those, for which this is possible, might need to be modified substantially to accommodate for the different scenarios.

Given the prominence of deep learning (DL) approaches in federated learning, we chose to explore a clustering model based on DL. The model we developed is based on the Deep Embedding for Clustering (DEC) model proposed by [10]. We made novel extensions to original DEC model in order to obtain a federated implementation, for extend to the MDS problem, and for study its generative capabilities. The federated implementation was developed using the Flower framework [26]. In the following a complete explanation for our DEC model is provided. A description will be given for the many modifications we made both to the structure of the network and to the training procedure.

2.1.1 DEC model and our modifications

Original DEC model was inspired by parametric t-distributed stochastic neighbor embedding (t-SNE) and it exploited the ability of autoencoders to project real data to the feature, or hidden, subspace, with a (commonly) strongly reduced dimensionality. This model is able to simultaneously learn feature representations and cluster assignments, but we explored also its generative capabilities for producing synthetic data. The main problem is clustering a set of n points $\{x_i \in X\}_{i=1}^{n}$ into k clusters, each represented by a centroid μ_j, with$j = 1, \ldots, k$. Instead of clustering directly in the *data space* X, the model first transforms data with a non-linear mapping $f_\theta : X \longrightarrow Z$, where θ are learnable parameters and Z is the latent *feature space*. The dimension of Z is typically much smaller than X in order to avoid the "curse of dimensionality". In order to parametrize this mapping f_θ, deep neural networks are a natural choice. In this data driven approach, the optimization of this neural network is crucial to obtain reliable results. The model, in a second stage, trains the mapping f_θ coupled with an auxiliary distribution, that describes the distribution of data points' distances w.r.t. their estimated cluster center, minimizing the Kullback-Leibler (KL) divergence. This second step effectively executes the clustering by assigning probability labels to data points according to the auxiliary distribution, and it also optimizes at the same time the parameters θ and the clusters' centroids. In Fig. 2.1, we propose the original image representing DEC model.

DEC training

DEC training is non trivial and subdivided in two main steps:

Step 1. Parameters initialization and feature space identification by training a deep autoencoder.

Step 2. Parameters optimization, i.e. clustering step, exploiting an auxiliary target distribution.

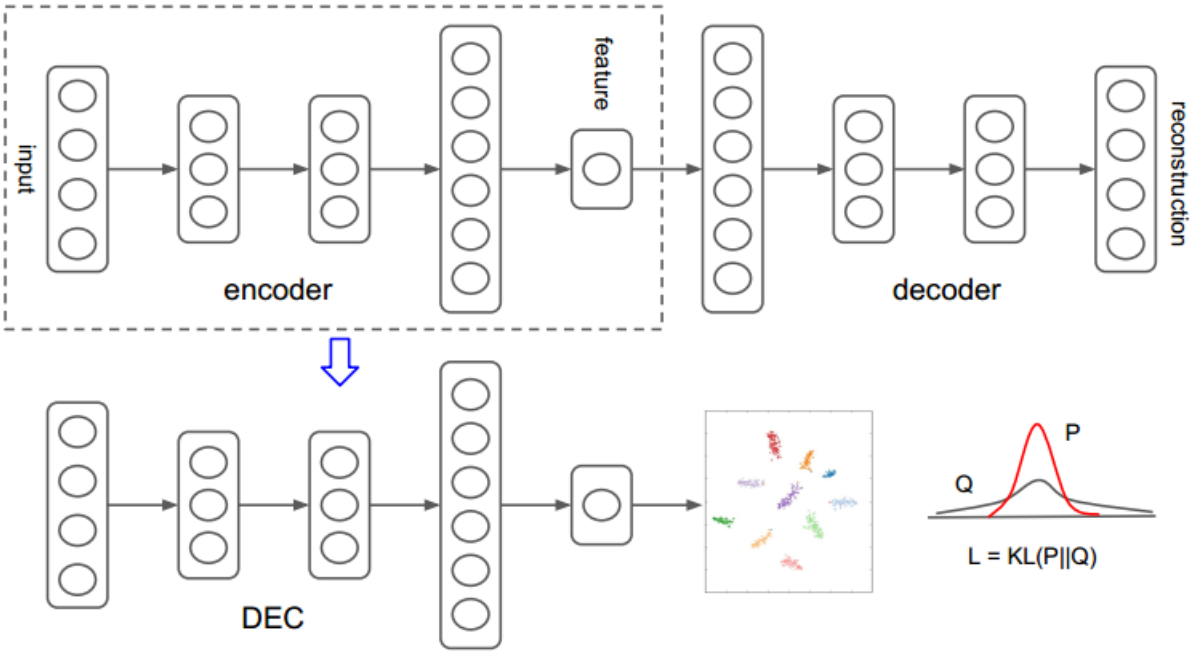


Figure 2.1: Diagram of the DEC model from the original paper.

During Step 1, the model tries to learn the feature representation of data, i.e. learns the non-linear mapping f_θ. A stacked autoencoder (SAE) was used by the authors because recent research has shown that these consistently produce semantically meaningful and well-separated representations on real-world data sets. SAEs are deep neural networks (DNN) composed by many stacked densely connected linear layers that can be separated in half by identifying an "encoding" stack of layers and a "decoding" stack. The dimensions of the layers in these two stacks are usually the same but in the inverse order producing a bottleneck in the middle of the structure that represents the feature space. In this architecture, the output of all the hidden layers is activated by a non-linear function, except for the bottleneck layer representing feature space. Fig. 2.2 shows a diagram of such architecture.

Thus the unsupervised representation of the feature space Z, learnt by SAE, facilitates in a natural way the clustering representation from DEC model. In order to obtain the best representation possible, the SAE initially undergoes a pretraining where it initializes its parameters and then a finetuning. Pretraining was performed in a greedy-layer wise (GLW) fashion, i.e. a series of coupled denoising autoencoders (DAE) composed of two

layers only was trained sequentially. DAEs are two layer neural networks defined by:

$$\tilde{x} \sim \textit{Dropout(x)} \tag{2.1}$$
$$h = g_1 W_1 \tilde{x} + b_1 \tag{2.2}$$
$$\tilde{h} \sim \textit{Dropout(h)} \tag{2.3}$$
$$y = g_2 W_2 \tilde{h} + b_2 \tag{2.4}$$

where $\textit{Dropout}(\cdot)$ is a stochastic mapping that randomly sets a portion of its input dimensions to 0, g_1 and g_2 are activation functions for encoding and decoding layers respectively, and $\theta = \{W_1, b_1, W_2, b_2\}$ are model parameters. The objective of using DAEs is to eliminate the risk of learning the so-called "identity function". Usually the reconstruction loss to minimize is the Mean-Squared Error (MSE), optimal for real-valued data, but it has shown reliable results also for other kinds of data.

$$\text{MSE} = \frac{\sum_{i=1}^{n}(y_i - x_i)^2}{n} \tag{2.5}$$

These networks are optimized using the Stochastic Gradient Descent (SGD) optimizer. During finetuning, dropout layers are removed and the coupled pretrained DAEs are stacked together composing a SAE, which is then trained optimizing again the reconstruction loss. The final result of Step 1 is a multilayer deep autoencoder with a bottleneck encoding layer in the middle. After Step 1 is completed, the decoder's layers are discarded and only the encoder's layers will be used as initial mapping between the data space and the feature space.

I explored a less complex and more reliable training procedure exploiting tied stacked autoencoders (TSAE), whose main difference w.r.t. SAEs is that the parameters composing the decoder's layers are tied to those of the encoder's layers through a transposition operation. The Fig. 2.3 shows a diagram of such architecture.

I decided also to test the addition of some hidden dropout layers, that is a very effective technique for avoiding overfit in this context [27]. Using properly initialized TSAE, e.g. by "Xavier-Glorot" initialization [28], results in many advantages. They are explained in the following.

- Thanks to the tying of layers' parameters, the actual number of parameters to train is halved, meaning, often, faster training. On the other hand the complexity of the mapping f_θ is conserved and not reduced.

- The training procedure does not need to perform a pretraining step in the GLW fashion. GLW training is, in general, power and time consuming since the coupled DAEs must be trained in sequence for many epochs on the entire data set. This reasoning is even more valid in a federated setting where convergence is usually slower, therefore more training epochs are needed.

- This architecture allows to test modifications to the original loss function for taking into account the nature of data. One may wonder why this was not possible before, in fact, technically, one could use different losses in the finetuning of the SAE, but not in GLW pretraining. This is because the nature of the hidden representation of data given by the coupled DAEs cannot retain the structure of the data, i.e. if data is binary, its hidden representation from any DAEs, after having stacked it in a SAE, will not be. On the other hand, applying different losses during GLW pretraining and finetuning does not carry many advantages, since GLW pretraining works as an initializer constraining strongly θ parameters.
- This architecture make use of modern optimization algorithms, e.g. Adam [29], Yogi [30], these have never been experimented in GLW pretraining coupled with finetuning.
- This tied architecture allows us to study the generative capabilities of the model after the clustering step. Since in the next step the decoder will not be trained at all, it may lose its property of back-projection from feature to data space. In our case, being its parameters tied to those of the encoder, that is further optimized, decoder may be able to retain the mapping even after the following step.

Optimizers

Since the modifications in architecture we made, we decided to experiment also on different optimizers for training the TSAE. The architecture proposed exploits some modern artifacts of computer science, and therefore exploiting modern artifact also in the optimization seemed reasonable. We chose to experiment SGD, as used in the original DEC, but also Adam and Yogi. These are modern adaptive optimization algorithms built on the basis of SGD. In the following we will give a brief explanation for these optimizers.

Stochastic Gradient Descent. This is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data, i.e. a batch). Especially in high-dimensional optimization problems this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate. Stochastic gradient descent has become one of the most popular optimization methods in machine learning. Its iterative procedure can be summarized in the following equation:

$$w_{t+1} := w_t - \eta \nabla Q_i(w_t) \tag{2.6}$$

where w_{t+1} are the parameters of the NN model at the next iteration $t+1$, η is the learning rate, $\nabla Q_i(w_t)$ is the gradient of the loss function $Q(w)$ computed

on the parameter w_t at current iteration t on the i^{th} sample. It is often used "with momentum", meaning that the current update of the parameter is linearly combined with the previous updates. This can be summarized by the following equations:

$$\Delta w_t := \alpha \Delta w_\tau - \eta \nabla Q_i(w_t) \qquad\qquad w_{t+1} := w_t + \Delta w_t \tag{2.7}$$

where Δw_τ represents the contribution of earlier updates, and α is an exponential decay factor between 0 and 1 that determines the relative contribution of the current gradient and earlier gradients to the parameters change. A suitable value for α is 0.9.

Adam. Adaptive Moment Estimation (Adam) is an update to the Root Mean Square Propagation optimizer. In this optimization algorithm, running averages of both the gradients and the second moments of the gradients are used. Given parameters $w^{(t)}$ and a loss function $Q^{(t)} = Q(w^{(t)})$, where t refers to the current training iteration, Adam's parameter update is given by:

$$\hat{m}_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w Q^{(t)} \tag{2.8}$$

$$\hat{v}_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w Q^{(t)})^2 \tag{2.9}$$

$$\hat{m}_w = \frac{\hat{m}_w^{(t+1)}}{1-\beta_1^t} \tag{2.10}$$

$$\hat{v}_w = \frac{\hat{v}_w^{(t+1)}}{1-\beta_2^t} \tag{2.11}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w + \epsilon}} \tag{2.12}$$

where $\epsilon > 0$ is a small scalar, that we set to 10^{-8}, used to prevent division by 0, and $\beta_1 \geq 0$, that we set to 0.9, and $\beta_2 \leq 1$, that we set to 0.999, are the forgetting factors for gradients and second moments of gradients, respectively. Squaring and square-rooting is done element-wise.

Yogi. This is essentially a modification of Adam algorithm. The key element behind Adam is to use an adaptive gradient while ensuring that the learning rate does not decay quickly. To achieve this, Adam uses an Exponetial Moving Average (EMA) which is, by nature, multiplicative. This leads to a situation where the past gradients are forgotten in a fairly fast manner. This can especially be problematic in sparse settings where gradients are rarely nonzero. An alternate approach to reach the same results as Adam is through additive updates. To this end, a simple additive adaptive method was proposed in [30], Yogi (name derived from the Sanskrit word "yuj" meaning "to add"), to improve the stochastic nonconvex optimization

problem of our interest. Given parameters $w^{(t)}$ and a loss function $Q^{(t)} = Q(w^{(t)})$, where t indexes the current training iteration, Yogi's parameter update is given by:

$$\hat{m}_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w Q^{(t)} \tag{2.13}$$

$$\hat{v}_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)\cdot \text{sign}[\hat{v}_w^{(t)} - (\nabla_w Q^{(t)})^2]\cdot(\nabla_w Q^{(t)})^2 \tag{2.14}$$

$$\hat{m}_w = \frac{\hat{m}_w^{(t+1)}}{1-\beta_1^t} \tag{2.15}$$

$$\hat{v}_w = \frac{\hat{v}_w^{(t+1)}}{1-\beta_2^t} \tag{2.16}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w + \epsilon}} \tag{2.17}$$

where $\epsilon > 0$ is a small scalar, that we set to 10^{-8}, used to prevent division by 0, and $\beta_1 > 0$, that we set to 0.9, and $\beta_2 < 1$, that we set to 0.999, are the forgetting factors for gradients and second moments of gradients, respectively.

Clustering step

Step 2 is then performed to refine the parameters found so far. The data is passed through the initialized encoder to get embedded data points and then K-means clustering is performed in the feature space Z to obtain k initial centroids $\mu_j, j = 1, \ldots, k$. We explored, in addition to the original procedure, the possibility to re-scale the points in the feature space, before applying K-means clustering, in order to help the algorithm understand better the clusters' distribution. Whenever feature data points have been re-scaled, we used the empirical centroids computed on the original feature space from the labels obtained from K-means to initialize the centroids.

Following to this initialization, the training proceeds iterating between two tasks: computing the auxiliary target distribution, and minimizing the Kullback-Leibler divergence (KLD) loss between the predicted target distribution and the computed target distribution. We chose the KLD because it describes the information lost by approximating a distribution with an empirical one. The auxiliary distribution is the kernel of the DEC model that measures the similarity between embedded points and centroids. The choice of the Student's t-distribution seemed natural (including the connection to the t-SNE encoding). This target distribution is computed as:

$$q_{ij} = \frac{(1 + ||z_i - \mu_j||^2/\alpha)^{\frac{\alpha+1}{2}}}{\sum_{j'}(1 + ||z_i - \mu_{j'}||^2/\alpha)^{\frac{\alpha+1}{2}}} \tag{2.18}$$

where $z_i = f_\theta(x_i) \in Z$ corresponds to $x_i \in X$ after embedding, α represents the degrees of freedom of the Student's t-distribution that is usually proportional to the number of

observed clusters minus one, and q_{ij} can be interpreted as the probability of assigning sample i to cluster j (i.e. a soft assignment). Originally, the authors of DEC model let $\alpha = 1$, stating that it is not possible to cross-validate α on a validation set in an unsupervised setting, and learning it is superfluous. In the article that inspired DEC model, [31], they suggest, however, to try to learn parameter α from data, and they say also that the best performing value is close to $\alpha = N - 1$, where N is the number of clusters.

For an efficient convergence of the model, choosing the proper target distribution P is very important. It should possess the following properties.

1. Strengthen predictions (i.e. improve cluster purity).
2. Put more emphasis on data points assigned with high confidence.
3. Normalize loss contribution of each centroid to prevent large clusters from distorting the hidden feature space.

During the KLD optimization, the model updates the deep mapping and refines the cluster centroids by learning from current high confidence assignments using the computed auxiliary target distribution. Specifically, the model is trained by matching the soft assignment to the target distribution. The objective is defined as a KL divergence loss between the soft assignments q_i and the auxiliary distribution p_i:

$$L = \mathrm{KL}(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.19}$$

where p_{ij} are computed as:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij}^2 / f_{j'}} \tag{2.20}$$

where $f_j = \sum_j q_{ij}$ are the computed (soft) cluster frequencies.

The method jointly optimizes cluster centers and encoder's parameters θ. It originally used Stochastic Gradient Descent (SDG) with momentum, so that the gradients of KLD were given by:

$$\frac{\partial L}{\partial z_i} = \frac{\alpha + 1}{\alpha} \sum_j \left(1 + \frac{||z_i - \mu_j||^2}{\alpha}\right)^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j), \tag{2.21}$$

$$\frac{\partial L}{\partial \mu_j} = -\frac{\alpha + 1}{\alpha} \sum_i \left(1 + \frac{||z_i - \mu_j||^2}{\alpha}\right)^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j). \tag{2.22}$$

We explored also the possibility to use Adam and Yogi as well for this optimization step. The gradients $\frac{\partial L}{\partial \theta}$ are then passed down to the encoder and used in standard

backpropagation to compute gradient for the encoder's parameter $\frac{\partial L}{\partial \theta}$. For the purpose of discovering cluster assignments, the procedure is stopped when less than a pre-defined number of points change cluster assignment between two consecutive iterations, usually this threshold is set to 0.1%. This training strategy for the clustering step can be seen as a form of self-training [32]. As in self-training, we take an initial classifier and an unlabeled data set, then we label the data set with the classifier in order to train on its own high confidence predictions. Indeed, in experiments the authors observed that DEC improves upon the initial estimate in each iteration by learning from high confidence predictions, which in turn helps to improve low confidence ones.

2.1.2 Federated training of our DEC model

Training our DEC model in a federated fashion needs some modifications w.r.t. the centralized version. Both step 1 of training (initialization of parameters via TSAE) and the minimization of KLD loss are optimized by exploiting FedAvg [19], i.e. the most used algorithm for aggregating weights, when the local optimizer is SGD, FedAdam [25] when Adam is used as local optimizer, FedYogi [25] when Yogi is used as local optimizer. The computations involved in FedAdam and FedYogi are resumed by the pseudocode in Algorithm 2. The federated implementation of these optimizers assumes the choice of an additional set of hyperparameters (η_s, β_{1s}, β_{2s}) referring to server aggregation only. The local estimate of gradients could be performed, in principle, with any local optimizer.

Algorithm 2: **FedAdam**, and **FedYogi** algorithms.

Initialization:

x_0, $v_{-1} \geq \tau^2$, server learning rate η_s, server decay parameters $\beta_{1s}, \beta_{2s} \in [0, 1)$, client learning rate η_l

for $t = 0, ..., T$ **do**

  Sample subset $\mathcal{S}$ of clients

  Set $x_{i,0}^t = x_t$

  for *each client* i *in* $\mathcal{S}$ **do in parallel**

    for $k = 0, ..., K-1$ **do**

      Compute an unbiased estimate $g_{i,k}^t$ of $\nabla F_i(x_{i,k}^t)$

      $x_{i,k+1}^t = x_{i,k}^t - \eta_l g_{i,k}^t$

    end

    $\Delta_i^t = x_{i,K}^t - x_t$

  end

  $\Delta_i = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \Delta_i^t$

  $m_t = \beta_{1s} m_{t-1} + (1 - \beta_{1s}) \Delta_t$

  $v_t = v_{t-1} - (1 - \beta_{2s}) \Delta_t^2 \text{sign}(v_{t-1} - \Delta_t^2)$ **(FedYogi)**

  $v_t = \beta_{1s} v_{t-1} - (1 - \beta_{2s}) \Delta_t^2 \Delta_t^2$ **(FedAdam)**

  $x_{t+1} = x_t + \eta_s \frac{m_t}{\sqrt{v_t} + \tau}$

end

Federated aggregation of initial centroids

The initialization of the cluster centroids needs a deeper discussion, in fact it is not guaranteed that running K-means at clients's place will uncover the same clusters centroids for every client. One has to keep in mind that clusters' centroids initialization does not need to be extremely precise because centroids will be optimized during KLD minimization step, but on the other hand they cannot be random since DEC model performs better if clusters are already enough separated in the feature space. In order to solve the problem of "federating" the clusters' centroids initialization, we have proposed four different solutions.

The first solution proposed is an algorithmic procedure described in the following lines. Every client identifies its clusters' centroids similarly to the centralized implementation. Those are then sent to the central node to be aggregated. The server, thus, receives N set of k centroids, where N is the number of clients, and k is the number of centroids that locally, i.e. at client place, K-means algorithm has searched for. The aggregation of centroids is performed "by completion" as follows:

1. one of the N sets of client's centroids is chosen randomly, let's denote this client with letter "i" s.t. its set of centroids is denoted by $\{\mu^i\}_{j=1,...,k} = \{\mu_1^i, ..., \mu_k^i\}$;

2. the first of the k centroids, μ_1^i, in the set is chosen to initialize the set of aggregated centroids, denoted with letter "s": $\{\mu^s\}_{j=1,...,k} = \{\mu_1^s, ..., \mu_k^s\}$, s.t. $\mu_1^s = \mu_1^i$;

3. the other $[k \cdot (N-1)]$ centroids, those of the clients that are not chosen, along with the $(k-1)$ centroids of the client chosen constitute the set of $(k \cdot N - 1)$ centroids, denoted by letter "c" $\{\mu^c\}_{z=1,...,(kN-1)} = \{\mu_1^c, ..., \mu_{(kN-1)}^c\}$, that will be chosen for completing the set of aggregated centroids $\{\mu^s\}_{j=1,...,k}$ initialized by $\mu_1^s = \mu_1^i$;

4. for every centroid in $\{\mu^c\}_{z=1,...,(kN-1)}$ is computed the euclidean distance w.r.t all the centroids in $\{\mu^s\}_{j=1,...,k}$ and only the minimum value is considered, i.e. a set of distances $\{d^c\}_{z=1,...,(kN-1)}$ is computed s.t. $d_z^c = \min_{j=1,...,k} \|\mu_z^c - \mu_j^s\|$;

5. the centroid from $\{\mu^c\}_{z=1,...,(kN-1)}$ that has the maximum distance d_z^c is then added to $\{\mu^s\}_{j=1,...,k}$;

6. steps 4 and 5 are iterated until the set $\{\mu^s\}_{j=1,...,k}$ is completed with k centroids.

In this procedure we are assuming that the distance between a centroid and a set of centroids could be estimated by getting the minimum between the euclidean distances of this centroid w.r.t. the centroids in the set. The meaning of the procedure of adding up to the set of aggregated centroids the centroid that has the maximum distances w.r.t. the set is that we are trying to take into account the spread of the clusters over the entire feature space. The the final list of aggregated centroids is sent then back to clients to initialize the local clustering layer. We can call this procedure "max min".

Another simpler technique to aggregate those centroids could be the random sampling of k centroids from the shuffled list of all the $N \cdot k$ centroids. We call this procedure "random".

The previous random sampling could be weighted in probability by the number of data points belonging to centroids' clusters. We call this procedure "random weighted". In this last case, clients should return to server couples (μ_j^i, n_j^i), where letter i indicates the i^{th} clients and letter j the j^{th} cluster.

Another interesting aggregation procedure we developed exploits, as before, the couples (μ_j^i, n_j^i), but these are used to build a dummy "simulated" feature space. Every centroid μ_j^i is repeated once for every n_j^i, building a feature space composed of repeated centroids. This stage could be enhanced by building the feature space with repeated $\mu_j^i + \mathcal{G}_i^j$, where $\mathcal{G}_i^j$ is small additive Gaussian noise, but we have not tested this variant. Then, once the feature space of centroids has been built, another round of K-means clustering is performed by the server. The centroids found by K-means algorithm compose the final list of aggregated centroids that is sent then back to clients to initialize the clustering layer. We call this method "double kmeans".

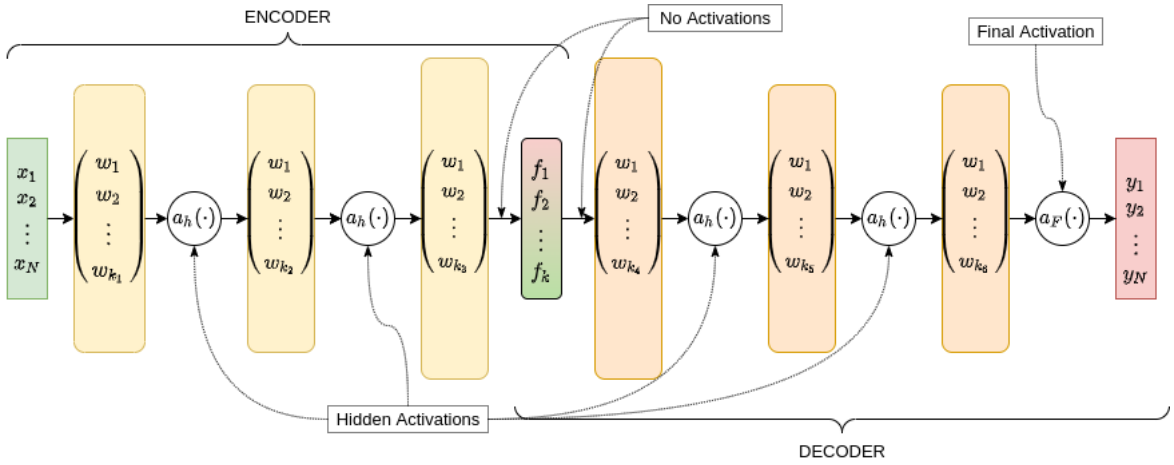


Figure 2.2: Diagram of a SAE with 3 hidden layers for the encoder and for the decoder stacks.

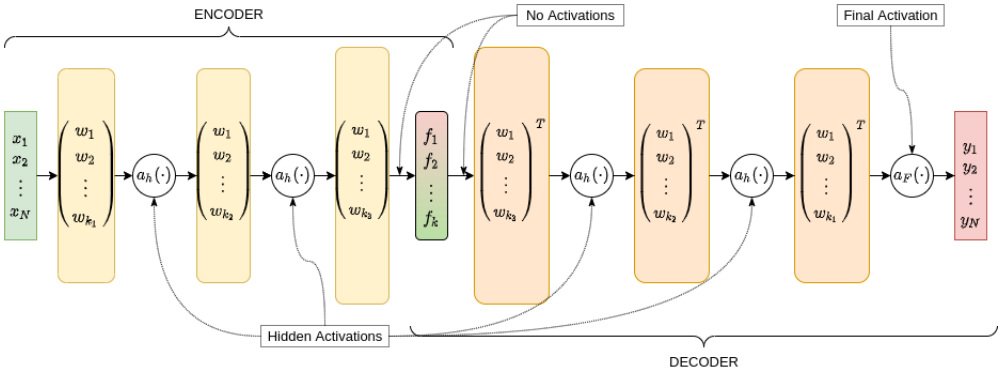


Figure 2.3: Diagram of a TSAE with 3 hidden layers for the encoder and for the decoder stacks.

Chapter 3

Data

In this section we will discuss the data sets used in this work. In particular we will describe those data sets used as a baseline to prepare the way for training our DEC model on EUROMDS data set. We will describe also EUROMDS data set explaining which variables and categories we have used in this work. We chose MNIST data set as a baseline, for the following reasons.

- We were able to perform the clustering analysis having available actual labels from which we could compute the accuracy of our DEC model.
- It was used by the author of original DEC, thus we had the option to directly compare the performance of our DEC model w.r.t. the original one.
- With an artificial rounding operation, we were able to build a binarized version of MNIST, that thus had the same nature of EUROMDS data set.

The final objective was to solve the task of unsupervised clustering on the EUROMDS data set which is composed of, essentially, binary data, and that is why it was important to have a baseline data set that was similar to that in nature. Moreover, since we have made important modifications to the original DEC model, some of them in order to exploit newer neural network architectures, to manage the binary nature of our data, and to enable the possibility to use it in a federated setting, it was really important to test our DEC model performance w.r.t. original DEC. Having a performance comparison w.r.t. the original DEC model was important to understand if our choices could increase or not the performance.

3.1 (B)MNIST

MNIST data set is a very popular collection of images, a complete description is available at [33]. It is composed of a training set of 60,000 examples, and a test set of 10,000

examples. These are gray level images of 28×28 pixels representing handwritten digits from 0 to 9, and thus composed by 10 classes. The digits have been size-normalized and centered in a fixed-size image. The sample population is uniform w.r.t. the classes.

In order to have a baseline data set that could be comparable to the EUROMDS binary data set, we chose to binarize those images. Each image pixel is then scaled in the range $[0, 1]$, since the values of pixels' gray levels are provided in the range $[0, 255]$. Then they are rounded to assume values 0 or 1 with the usual rounding threshold of 0.5. The transformed data set will be then denoted as Binary MNIST (BMNIST). An example of this is shown by Fig. 3.1

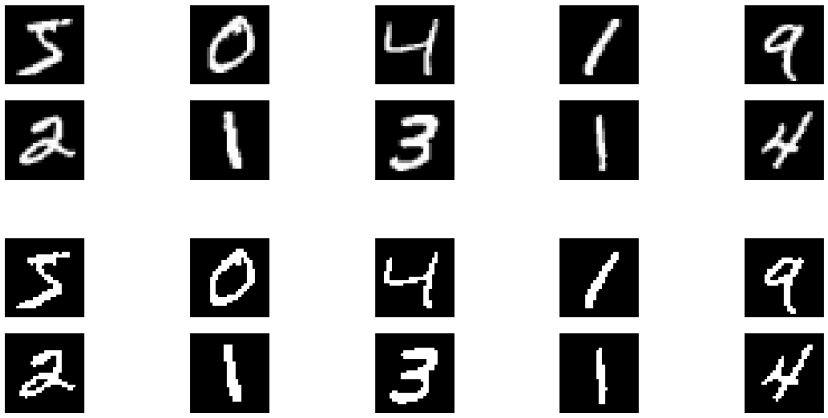


Figure 3.1: This figure shows the transformation from classical MNIST images to BMNIST images. The upper ten images were sampled from MNIST and then transformed in their binarized version represented by the lower ten images.

3.2 MDS data description

The data on Myelodisplastic Syndrome (MDS) that are actually available between the GenoMed4ALL [34, 35] consortium are a combination of genomic, cytogenetic and clinical data. GenoMed4ALL is an EU-funded project that aims to accumulate evidence points towards personalised medicine to treat and manage common, rare and ultra-rare haematological diseases. To address this need and support research further, the project is developing a data-sharing platform that utilises novel advanced AI models. The project's partners are sharing infrastructure, powerful computing facilities, hospital registries and data processing tools towards this effort. The project is contributing to the exploitation of omics and clinical data in patient-oriented research and decision-making through advanced statistics and machine learning approaches. These tools are developed in the hope to obtain improved therapies and better clinical outcomes.

I will report here a brief description of EUROMDS data set. The data set is in a "csv" format and we have that rows represent patients while the columns are the variables (features). These variables are organized as follows:

- **General and demographic variables**
  - Patient ID, EUROMDS patient labels
  - Gender, Male or Female
  - Age at data collection, age at diagnosis
- **Clinical variables (prognostic scores)**
  - World Health Organization (WHO) 2016 subtype, WHO disease classification
  - International Prognostic Scoring System (IPSS) risk group, disease classification according to IPSS
  - Revised International Prognostic Scoring System (IPSSR) risk group, disease classification according to IPSSR
- **Clinical-biological variables at diagnosis**
  These variables are composed of haematochemical tests results on blood (cell counts and ferritin etc.) and bone marrow (count of bone marrow blasts an sideroblasts etc.) and a comorbidity score index.
- **Follow up and outcome variables**
  - Leukemia free survival, characterized by two variable: event, and time to event
  - Overall survival: event, and time to event
  - Acute Myeloid Leukemia (AML) adjusted overall survival: event, and time to event
- **Cytogenetic variables**
  These data are grouped in 13 columns where is reported the presence/absence of a chromosomal alteration. This data is represented by 0 when chromosomal alteration absent, 1 when chromosomal alteration is present, "NaN" if not measured.
- **Genomic variables**
  These variables are composed by the mutational status of 47 selected genes (gene panel for MDS). This data is represented by 0 where mutation is absent, 1 where mutation is present, "NaN" if not measured.

- **HDP components**
  Patient specific Hierarchical Dirichlet Processes (HDP). The weights across 6 latent components are listed as reported in [14].

In this work only a part of these variables were used: from the sets of genomic variables, and cytogenetic variables were extracted those variables for training our DEC model, precisely the same used in [14] for HDP; follow up and outcome variables were used to characterize the clusters obtained by our DEC model; HDP components were used as "ground truth" to evaluate our DEC model clustering results. Since the columns of variables used in the end were 54, we decided to give simple representation of a patient as an image 6×9 were every pixel corresponds to a specific variable as described by Table 3.1. These kind of visual representations are often helpful for clinicians because they are able to recognize features with a look. An example of this representation is given by Fig. 3.2.

ICHR17qort17p	IDH2	ETV6	PIGA	SF3B1	ATRX
IDH1	NOTCH1	LOCHR13OD13q	ASXL1	FLT3	BCOR
LOCHR5OD5qPLUSother	BCORL1	TET2	ZRSR2	DNMT3A	LOCHR12OD12P12p
RUNX1	SMC1A	LOCHR20OD20q	MPL	BRAF	idicXq13
NRAS	NF1	SMC3	PHF6	LOCHR7OD7q	LOCHR9OD9q
EZH2	U2AF1	GNAS	WT1	GNB1	RAD21
SRSF2	TP53	CBL	KRAS	FBXW7	del5q
LOCHR11OD11q	PTPN11	GATA2	KIT	NPM1	GOCHR8
JAK2	LOCHRY	PRPF40B	CEBPA	CBLB	STAG2

Table 3.1: This table represents how the variables used in this work are arranged as an image. Each cell corresponds to one pixel of the image, while rows and columns arrange those pixels to form an image. Mutations are represented with their acronym.

Figure 3.2: Here are shown 5 examples of the image representation of MDS patients. These are reconstructed synthetic data, and thus do not represent real patients. White pixels represent the value 1, while black pixels value 0. Gray values represent probabilities of having 1.

In order to fill those values where the measure could not be accomplished, and thus represented by "NaN", we decided to replace "NaN" with the empirical probability of having value 1 in that column. This empirical probability was estimated by the frequency of 1s w.r.t. to all the patients that have that column filled with 1 or 0, i.e. excluding those with "NaN".

Chapter 4

Experiments

In this section we will present all the experiments we performed for clustering the data sets in the centralized approach. We will present at the end of the chapter the experiment performed on EUROMDS data set in the federated approach. The reference for clustering MNIST has the best performance in accuracy from the original DEC, that is 84.30%. On the other hand, the reference for EUROMDS clustering is given by the hierarchical Dirichlet processes (HDP), a non parametric Bayesian approach to clustering grouped data, applied to this set in [14]. Since EUROMDS data set does not come with pre-assigned truth labels, HDP label assignments will be treated as "ground truth", i.e. our clustering evaluation will be in terms of these labels investigating the amount of information that our DEC model retains about HDP clustering. We will present also how we decided to evaluate the generative capabilities of the model, for which no reference exists, since the authors did not investigate these. It is important to say that it was not expected to achieve generative performances comparable to those of generative adversial networks (GANs), the "gold standard" for this task. On the other hand, we believed it was important to uncover potential generative capabilities of our DEC model, since producing synthetic data is an increasingly demanded task, especially in medical context.

4.1 Centralized approach

The experiments we will present first are those in the centralized approach. It often happens that a working centralized model is the starting point of a federated implementation, that is why we decided to find the best hyperparameters for a centralized training of our DEC model and then translate the implementation to the federated approach. These experiments will be divided in two parts: those referring to the optimization of the first step of DEC training, i.e. training the TSAE; and those referring to the optimization of other hyperparameters involved in the second step of DEC training, i.e. the actual clustering step.

4.1.1 TSAE hyperparameters

The goodness of training the TSAE is related to the value of reconstruction loss, the lower is the reconstruction loss, the better will be the training. A good training will guarantee a reliable construction of the feature space onto which the clustering is based. Having a minimal reconstruction loss would also help with the generative capabilities of our DEC model, since the clustering step corrupts partially, as we will see, the mappings of both encoder and decoder.

We tested then three network architectures composed of stacked fully connected tied layers with different dimensions. These will be listed in the following indicating the number of dimensions of the stacked layers of the encoder only, since the decoder has the same, but inverted, dimensions.

- "DEC" architecture has $[N, 500, 500, 2000, f_{DEC} = 10]$.
- "CURVES" architecture has $[N, 400, 200, 100, 50, 25, f_{CURVES} = 6]$.
- "GOOGLE" architecture has $[N, 1000, 500, 250, f_{GOOGLE} = 10]$.

Letter N represents the dimensionality of the input data space, and letter f the dimensionality of the feature space, that is fixed by the architecture. (B)MNIST data set has $N = 28 \times 28$, while EUROMDS data set has $N = 54$. The "CURVES" and "GOOGLE" architectures are those tested in [30], while "DEC" is the original one proposed by the authors. While "DEC" architecture was trained on every data set studied, "CURVES" and "GOOGLE" were trained only on EUROMDS data set.

We chose to test also different activation functions between the connected nodes. It is important to remember that the bottleneck is left without activation since it is used to represent the feature space. We differentiated the hidden activations between the stacked layers versus the output activation after the decoder. We chose to test rectified linear unit (ReLU) and sigmoid functions for both hidden and output activation.

We tested the insertion of hidden dropout layers with 6 different rates from 0.0, i.e. no dropout, to 0.5, i.e. half of the nodes are dropped out, with steps of 0.1. These were put before every hidden layer, excluded the one after the bottleneck.

We let the TSAE train for 500 epochs on (B)MNIST, while on EUROMDS for 150 epochs.

We set the batch size in training (B)MNIST to 256 samples, as the original DEC, while for EUROMDS we tested different values from 8 to 64, using powers of 2.

We tested three different optimizers: stochastic gradient descent (SGD), Adam, and Yogi. We tuned the learning rate for each optimizer in order to obtain the minimum loss. We tried also to exploit a learning rate scheduler, that reduces the learning rate value on the plateau of the loss.

Mean squared error was used as validation loss for the reconstruction error. We tested also a different training loss when training on EUROMDS that will be described in the

following. In order to take into account the binary nature of data, and at the same time preserving the reconstruction properties of TSAE, we tested a modification of the mean squared error loss. This is parametrized by a real value $\beta \in [0, 1]$ s.t. the modified loss function $\mathcal{L}_{\mathcal{F}}$ could be written as:

$$\mathcal{L}_{\mathcal{F}}(X, Y) = (1 - \beta) \cdot \|Y - X\|^2 + \beta \cdot \mathcal{F}(X, Y) \tag{4.1}$$

where X is the output of the TSAE, i.e. the input of the loss function, Y is the target of the loss function, i.e. the input of the TSAE since we are reconstructing, and $\mathcal{F}$ is a function applied to both output and target that modifies the loss. The function $\mathcal{F}$ was a combination of binary cross-entropy and dice coefficient:

$$\mathcal{F}(X, Y) = 1 - \frac{2|X \cap Y|}{|X| + |Y|} + \frac{1}{N} \sum_{j=1}^{N} \left[x_j \log y_j + (1 - x_j) \log(1 - y_j) \right] \tag{4.2}$$

where $X = (x_1, ..., x_N)$ and $Y = (y_1, ..., y_N)$.

We also tested two different procedures for noising the input data at training stage. The first one was the commonly used in denoising autoencoders, and consisted in corrupting some of the input data by setting part of the values to zero. The second one was the addition of random Gaussian noise to all the input data at every epoch. This additive Gaussian noise was centered on the value 0.5 with different values for σ. Noised input was finally truncated in the range $[0, 1]$ in order to keep parto of the binary nature of data.

4.1.2 DEC clustering hyperparameters

Once we had found the best values for the hyperparameters of TSAE training, we tested different setups for training the second step of our DEC model. The most important hyperparameter of the clustering step is the number of clusters k. For (B)MNIST this was set to 10 since we know there are 10 classes in the data set. For EUROMDS, relying on the HDP assignment where there are 6 classes, we chose $k = 6$.

The authors of DEC stated that this model is robust w.r.t. different values for the update interval λ, i.e. the number of batches to train before updating the auxiliary distribution. They set the update interval for MNIST to $\lambda = 160$ batch loops, i.e. they update the auxiliary distribution once every ~ 41000 samples since the batch size was set to 256. Their choice was made after having tested values for λ from the set $2^i \times 10$ for $i = 0, 1, ..., 8$. Since we made many modifications to the architecture of the initialization of parameters before the clustering step, we decided to leave out this parameter λ and to simply tune the batch size specific to the clustering step. Then the auxiliary distribution in our implementation was updated at every epoch. We also tested at this stage the three different optimizers, i.e. SGD, Adam, and Yogi, tuning the learning rate in order to have the best accuracy.

We tried to re-scale the feature space before running K-means algorithm, testing three different re-scale function. One was the standard scaler that exploits the common standardization procedure, i.e. it removes the mean value and standardizes the variance to unit norm. We tested also two normalizers, which scale independently the samples to unit norm. One of these used L1 norm and the other L2 norm.

We tested different values for the parameter α that describes the degrees of freedom of the auxiliary distribution. For (B)MNIST we tried $\alpha = 1$ as the authors of DEC, but also $\alpha = 9$ as suggested in [31]. For EUROMDS we tested, similarly, values of $\alpha = 1$ and $\alpha = k - 1$, where k is the number of clusters, initially set to 6.

In order to understand the goodness of the clustering for EUROMDS, for which ground truth labels there did not exist, we identified some metrics commonly used to score unsupervised clustering methods. The first metric we chose is the Silhouette Coefficient (Sil), that is calculated using the mean intra-cluster distance d_{in} and the mean nearest-cluster distance d_{near} for each sample as:

$$Sil = \frac{d_{near} - d_{in}}{\max(d_{near}, d_{in})} \tag{4.3}$$

Its value is constrained in $[-1, 1]$. We took the average between all the samples. The metric for computing the distance has to be chosen. We chose to compute the Sil for the data space using the cosine similarity metric (Cos), a popular metric for binary spaces, described by:

$$Cos(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^{N} A_i B_i}{\sqrt{\sum_{i=1}^{N} A_i^2}\sqrt{\sum_{i=1}^{N} B_i^2}}, \tag{4.4}$$

where A and B represent two elements of the data space, while for the feature space we chose the euclidean distance. The second score we chose is the Calinski Harabasz score (CH), also known as Variance Ratio Criterion, that is defined as the ratio between within-cluster dispersion and the between-cluster dispersion as

$$CH = \left[\frac{\sum_{k=1}^{K} n_k \|c_k - c\|^2}{K - 1}\right] / \left[\frac{\sum_{k=1}^{K} \sum_{i=1}^{n_k} \|d_i - c_k\|^2}{N - K}\right], \tag{4.5}$$

where n_k are the number of data points in the k^{th} cluster, c_k is the centroid of the k^{th} cluster, c is the global centroid, N is the total number of data points, and d_i is the i^{th} data point. I computed CH score for both data and feature space.

4.1.3 Generative capabilities of our DEC model

In order to evaluate the capabilities of our DEC model to generate synthetic data, we decided to study two metrics: the reconstruction loss at the end of our DEC model

training and the, as we called it, "cycle accuracy" that will be defined in the following. "Cycle accuracy" is defined as the accuracy of the prediction, given by our DEC model, on the real data w.r.t. the prediction on the reconstructed data, i.e. we let original data be reconstructed by the entire TSAE, i.e. encoder and decoder, and then we assign label prediction feeding our DEC model with reconstructed data. Reconstruction loss is the most important metric to understand the generative capabilities of the model since it essentially tells us how "good" the model is for producing simil-real data fed with real data. We assume the "cycle accuracy" to be informative on how consistent our model is, since it describes how much the model influences the features of the data it uses for producing labels, while eating data. The generation of synthetic data could be performed by our DEC model extracting the decoder from TSAE and feeding it with points in the feature space. This decoder fed with points of the feature space that are close to any centroid of any cluster, should produce synthetic data that is similar to the data our DEC model classified as belonging to that cluster.

4.2 Federated approach

The experiments using federated approach have the objective to understand how consistent the results of the federated model are w.r.t. the centralized one. These experiments have been conducted only using EUROMDS data set keeping as reference the "ground truth" labels from HDP, and also comparing federated results of federated DEC w.r.t. the centralized approach. We have tested different federated setups defined by different distributions of the samples to the clients. Two methods for distributing samples were used. The first one was to distribute uniformly the 2043 patients along a predefined number of clients C. An example of this distribution of samples is given by Fig. 4.1. The other method used to distribute the samples exploits an histogram, that followed

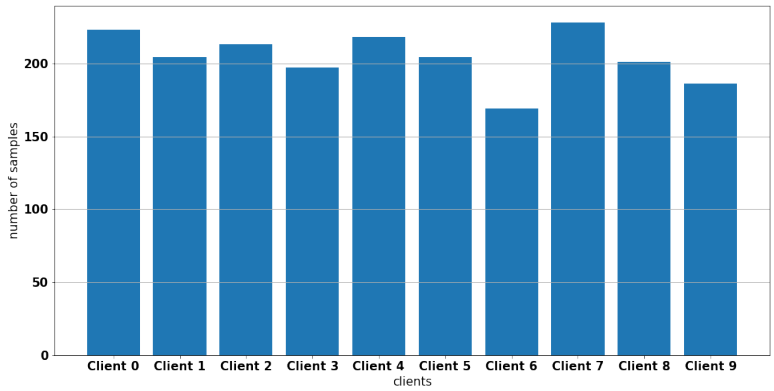


Figure 4.1: This histogram represents an example on how the 2043 samples from EUROMDS data set are being uniformly distributed between 10 clients.

a skewed Gaussian distribution, with a number of bins that was the number of clients. An example of this distribution of samples is given by Fig. 4.2. We chose to distribute

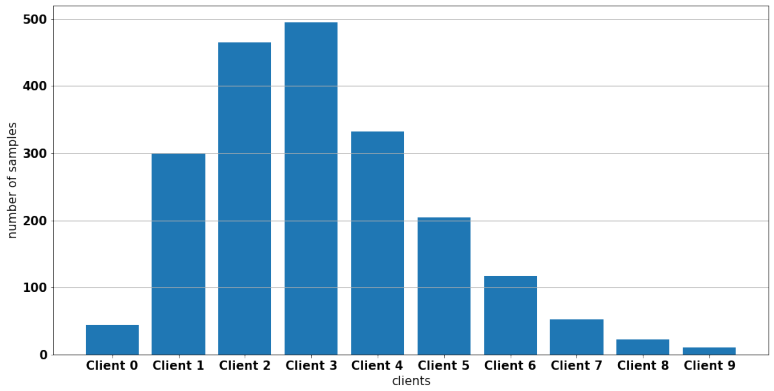


Figure 4.2: This histogram represents an example on how the 2043 samples from EUROMDS data set are being distributed between 10 clients following a skewed Gaussian distribution with skewness parameter equal to 4.

the data along $C = 10$ clients. We used both the two distributive methods described above. The hyperparameters chosen for the federated training of our DEC were derived from the centralized approach, this is the common procedure. We took then the best configurations for each optimizer using "DEC" architecture only. These 3 configurations were repeated for every distribution of samples and for every method of aggregation of centroids, i.e. 24 overall configurations.

Chapter 5

Results and Analysis

In this chapter we will present and discuss the results of the experiments we made. First the results on tuning the hyperparameters related to the training of TSAE will be presented. Then we will show the results on tuning the clustering step with the comparisons w.r.t. actual labels for (B)MNIST and HDP labels or clustering metrics for EUROMDS. At the end of the chapter, a deeper analysis on EUROMDS clustering results and the generative capabilities of our DEC model will be performed. For performing an efficient hyperparameter tuning, *Ray Tune* [36] was used along with Async Hyperband Successive Halving Algorithm (ASHA) [37]. The entire code is in *Python* exploiting the *PyTorch* [38] machine learning framework, everything will be available in a public repository [39] on *GitHub*.

5.1 TSAE hyperparameters

The very first hyperparameter tuned was the learning rate of training, for each of the three optimizes studied, i.e. SGD, Adam, and Yogi. Only "DEC" architecture was trained on both MNIST and BMNIST sampling 50 values for the learning rate from a log-uniform distribution between 10^{-6} and 1, seeking the lowest possible reconstruction loss. For this tuning, ReLU activation functions were used for both hidden activation and final activation, hidden dropout was not used, input data was not noised in any way, no modifications were applied to the reconstruction loss, and the number of epochs was reduced to 150. Another tuning with learning rate scheduler was performed to verify whether this procedure is useful or not. The same tuning was performed on EUROMDS, where we used all architectures, i.e. "DEC", "CURVES", and "GOOGLE". While for MNIST and BMNIST the batch size in the training of TSAE was fixed to 256, as the authors of DEC did, for EUROMDS it was set to 8 at this stage. Examples from MNIST tuning are shown in Fig. A.1 and in Fig. A.2, while examples from BMNIST tuning are shown in Fig. A.3 and in Fig. A.4. The other results, from EUROMDS, are shown in

the Appendix A. As expected, different optimizers perform better for different values of learning rate. The best values for each learning rate were annotated for the next steps, and they are tabulated in Appendix A. We also concluded that the use of a learning rate scheduler is superfluous and does not increase significantly the performance. Moreover, without using it, the final losses for different optimizers are closer to each other.

The next step was tuning the batch size, for EUROMDS training only. The values tested were 8, 16, 32, 64. At this stage, for every combination of optimizer, learning rate, architecture, we set these different values for the batch size and we tested at the same time all the possible sets of activations, i.e. ReLU and sigmoid functions as hidden and final activations. Results of this tuning are resumed in Tab. A.4 in Appendix A. The final values of the reconstruction loss for each configuration suggests that using different architectures does not change much the performance of the model at this step. Exception made for "CURVES" that performs in general slightly worse, "DEC" and "GOOGLE" perform very similarly. We concluded that this model applied to our problem is not more significantly dependent by the architecture of the TSAE. From these results, we annotated the best configurations for the next steps.

We then tested the use of hidden dropout with 10 different rates sampled uniformly in the range $[0.0, 0.5]$. Since training of TSAE did not show any sign of overfitting, we decided to test this regularizer only for EUROMDS. This tuning was made for the three optimizers along with the three network architectures for EUROMDS. Input data was not noised in any way, no modifications were applied to the reconstruction loss, and the number of epochs was reduced to 150. An example of this tuning is shown in Fig. 5.1, in Appendix A Tab. A.6 is resuming the best configurations. We decided not to use this regularizer because the best configurations resulted in a final loss function that was comparable to other configurations without hidden dropout. Moreover, the best configurations were often those with a minimal dropout rate. Regularizers as this one could however be useful for training on more populated data sets, or alternatively on data sets with more features than EUROMDS.

We also decided to test also another type of regularizer, i.e. noising the input data during the training step. We tested noising by corruption with 10 different rates of corruption sampled uniformly from the range $[0.0, 0.5]$. Here a percentage of the input data, specified by the corruption rate, is set to 0 before being fed to the network. Only EUROMDS data set underwent this experiment. An example of this tuning is shown in Fig. 5.2, in Appendix A Tab. A.5 is resuming the best configurations. We decided not to use this regularizer because the best configurations gave a final loss function that was comparable to other configurations without corruption. Moreover, the best configurations are often those with minimual value of corruption rate. This regularization is the most popular denoising procedure used in training neural networks. It is often useful for training on data sets that have a lot of features, only some of which are relevant to solve the problem. EUROMDS data set has very few features, thus we must consider all of them important for training.

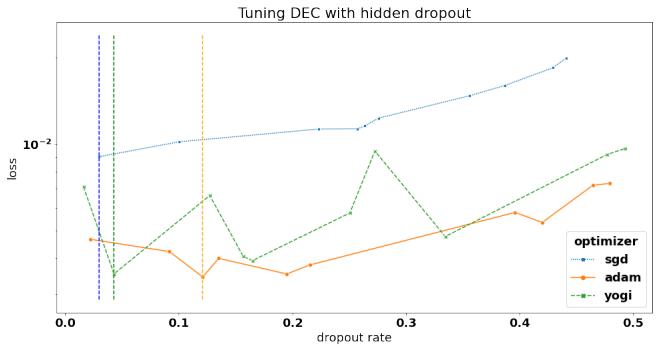


Figure 5.1: These are the results for training TSAE, using hidden dropout layers, with "DEC" architecture on EUROMDS data set with the three optimizers. Here, x axis represents the different values of rates used in the hidden dropout layers. The best configurations (minimal loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points.

We also tested noising with an additive Gaussian noise, as described in 4.1.1, centered in 0.5 with 10 different values of standard deviation σ sampled from the interval $[0.0, 0.2]$. When additive Gaussian noise was used, we decided to extend the training for another step of 150 epochs without applying the noise, as if treating the training with noise was like a parameter initialization. Only EUROMDS data set underwent this experiment. An example of this tuning is shown in Fig. 5.3, in Appendix A Tab. A.7 is resuming the best configurations. We decided not to use this regularizer because the best configurations give a final loss function that is comparable to other configuration without noise. Moreover, the best configurations are often those with minimal value of σ. This noising procedure could be useful for training on data sets whose features' values oscillate much more than in EUROMDS where, basically, data is composed of 0s or 1s. This can be viewed as an alternative denoising procedure.

We tested also the training procedures that used the modified loss, as described in 4.1.1. The parameter to set for these configurations was the value of β, i.e. the weighting factor for the modified loss. We sampled 10 values from the range $[0.0, 0.5]$. Only EUROMDS data set underwent this experiment. An example of this tuning is shown in Fig. 5.4, in Appendix A Tab. A.8 is resuming the best configurations. Incorporating data specific information in defining the loss could in principle represent a cornerstone when solving a specific problem. Studying which elements could be inserted in the loss function to this aim needs further in-depth discussion.

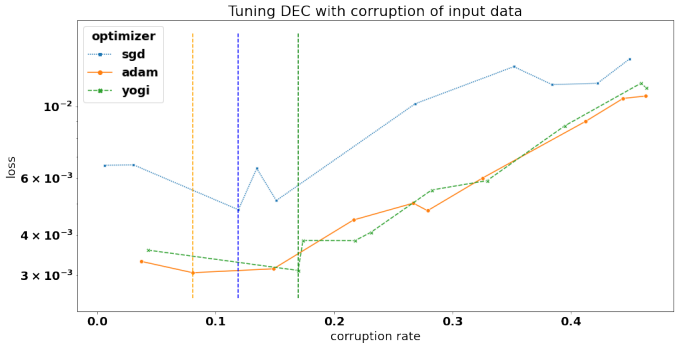


Figure 5.2: These are the results for training TSAE with "DEC" architecture on EUROMDS data set with the three optimizers where the input data has been corrupted as described above. Here, x axis represents the different values of corruption rate used. The best configurations (minimal loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points.

5.2 Clustering step hyperparameters

In order to set the hyperparameters of the clustering step, we decided to perform a grid search between the batch size and the learning rate. For this step we decided to keep the same optimizer used in the training of the TSAE. The values for the batch size tested were 64, 128, 256, 512 for MNIST and BMNIST data set, while for EUROMDS they were 8, 16, 32, 64. The 50 values searched for the learning rate were, as we did before when tuning the training of TSAE, sampled from a log-uniform distribution in the range $[10^{-6}, 1.0]$. For MNIST and BMNIST we chose to use "DEC" architecture only, while for EUROMDS all three architecture were tested. Then for MNIST and BMNIST we trained 600 configurations per data set, while for EUROMDS we trained 1800 configurations. The best configuration for MNIST and BMNIST data sets was chosen on the basis of the accuracy metric, since true labels were given. For the best configuration for EUROMDS data set an order of importance for the metrics was defined. The metrics were thus ordered as follows:

1. highest Silhouette Score computed in the data space, because we want the maximum similarity of real objects in their own clusters (cohesion in real data space) compared to other clusters (separation in real data space);

2. highest Calinski-Harabasz Score computed in the data space, for the same reasons as point 1;

3. highest Silhouette Score computed in the feature space, so that the structure of the feature space is considered but with lower importance w.r.t. the real data space;

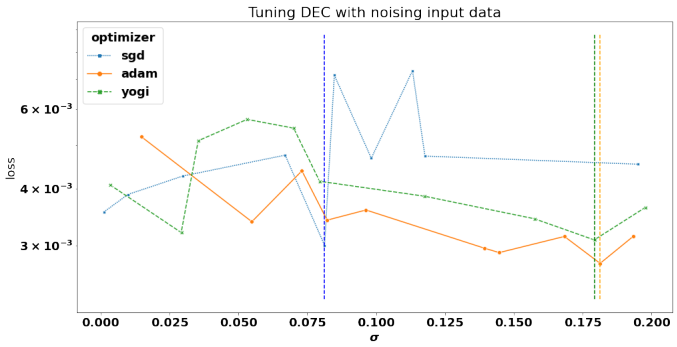


Figure 5.3: These are the results for training TSAE with "DEC" architecture on EUROMDS data set with the three optimizers where the input data has been noised as described above. Here, x axis represents the different values of σ used. The best configurations (minimal loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points.

4. highest Calinski-Harabasz Score computed in the feature space, for the same reasons as point 3;

5. lowest possible reconstruction error after clustering, to maintain the capability of our DEC model to be generative as high as possible.

The parameters of the best performing configuration were annotated to be used in the next studies. Results of this tuning are resumed in Appendix B. From Tab. B.1 to Tab. B.3, the results for MNIST are shown for every optimizer. BMNIST results are not reported since they were very similar to those of MNIST. Tab. B.6 and Tab. B.7 show the results about EUROMDS.

Once the best coupling of batch size and learning rate for every data set using every optimizer was found, we tested different values of α and tried using scalers. For MNIST and BMNIST we trained 24 other configurations for both data sets. EUROMDS did not undergo these tests. Complete tables of these results are available in Appendix B, Tab. B.4 for MNIST, and Tab. B.5 for BMNIST. The best performance in accuracy reached is 88.24% for MNIST, and 89.32% for BMNIST. These values are the evidence that our modifications to the original DEC model have accomplished better results and have improved the original model.

5.3 Analysis of MNIST/BMNIST clustering

At this stage I will present a deeper discussion about the clustering results on both MNIST and BMNIST data sets. The original DEC model was improved by our modifications resulting in higher accuracy. At the same time it is important to study the

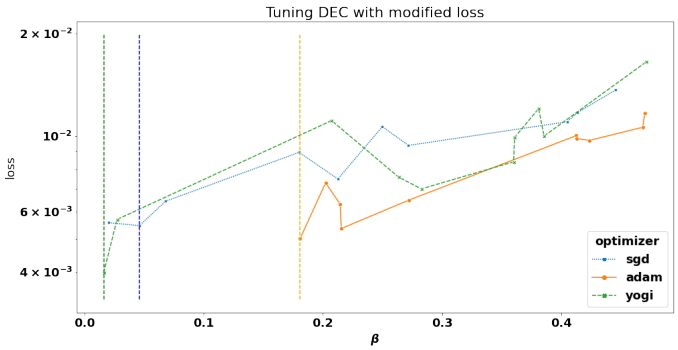


Figure 5.4: These are the results for training TSAE with "DEC" architecture on EUROMDS data set with the three optimizers which minimize the modified loss for different values of β. The best configurations (minimal loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points.

causes of failing to climb the wall of 90% accuracy. Studying the confusion matrix could help us to better understand how the accuracy is distributed along the labels, and also highlight if exist systematic errors in identifying specific labels. The confusion matrix in Fig. 5.5 shows that the model has a very good performance in identifying all the clusters exception made for 4s and 9s for which it performs very badly. The confusion matrix of the best configuration for BMNIST can be found in Appendix B, Fig. 5.6. The model cannot distinguish between 4s and 9s.

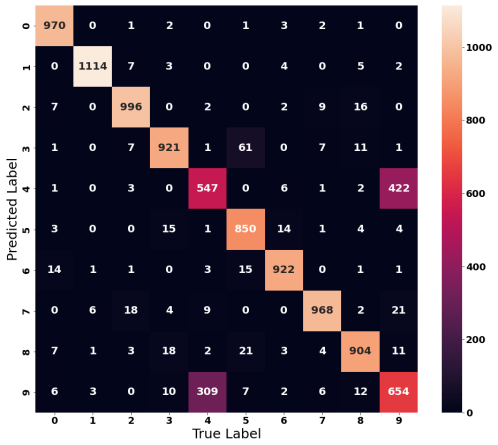
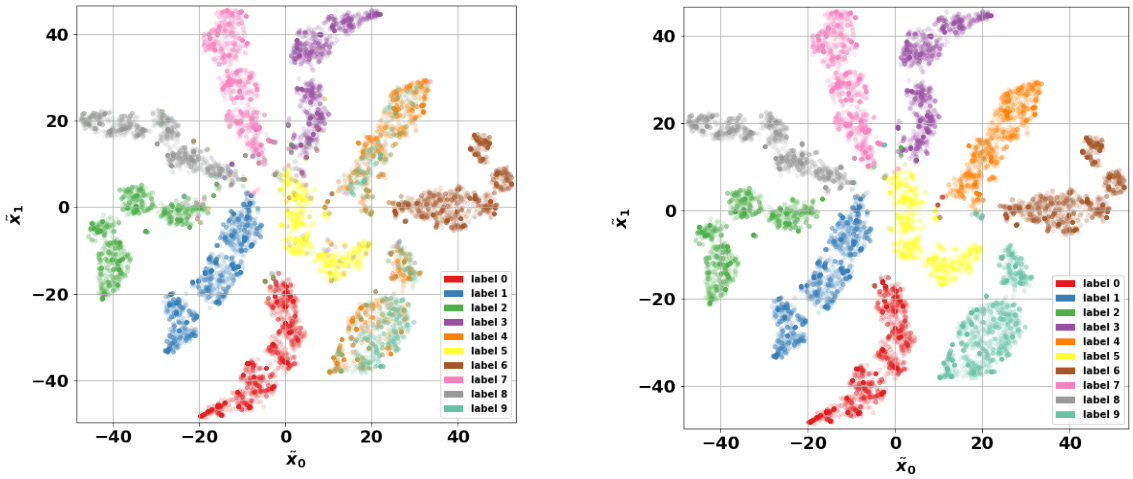


Figure 5.5: Confusion matrix for the accuracy of the best configuration of our DEC model on MNIST data set.

A reduced 2-D representation of the feature space shows that two distinct clouds are identifiable, but 4s and 9s overlap in both clouds. In order to represent the feature space

in a 2-D plane we decided to use the t-SNE reduction as DEC model took inspiration from it. A closer look to these clouds suggests that the model is able to distinguish 4s and 9s that have a diagonal main segment from those that have a vertical main segment, as Fig. 5.7 shows. But the same time it is not able to discriminate between 4s and 9s.



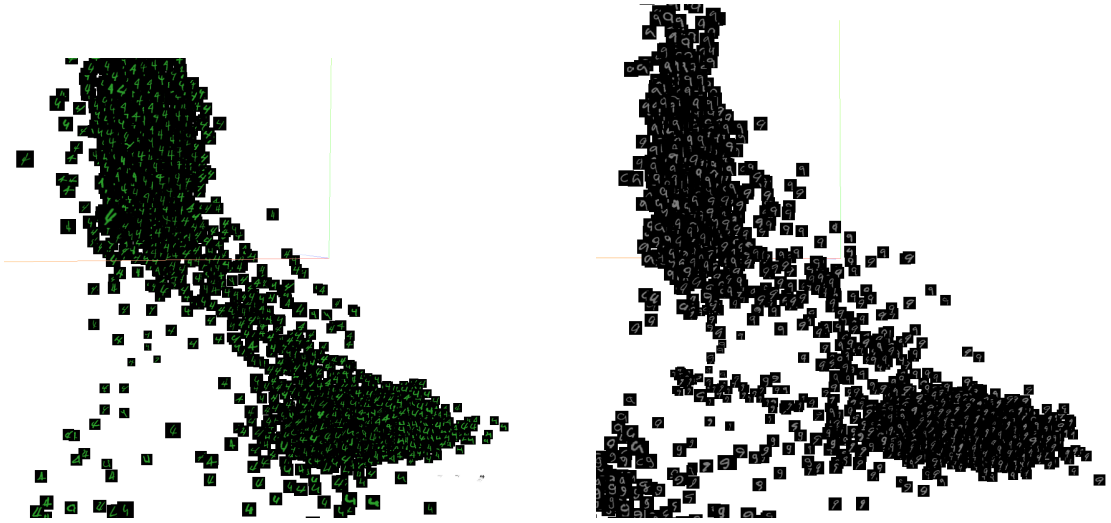
(a) (b)

Figure 5.6: This figure represents the 2-D reduction, performed using t-SNE, of the feature space at the end of the clustering step for the best performing configuration of our DEC model. On the left (a) points are labelled by actual labels, while on the right (b) by predicted labels.

Looking at 2-D representation of the feature space of different clustering steps, we are able to understand how the model separates clusters in the feature space. Consecutive clustering step iterations are condensing the cloud representing a cluster towards its centroid, this is also refined by the optimizer. This action explains why the silhouette score and the CH score in the feature space assume such high values. An example of this behaviour is shown by Fig. 5.8.

5.4 Analysis of EUROMDS clustering

Similarly as done before, the reduced 2-D representation of the feature space shows us the structure built the TSAE. Fig. 5.9 shows two reduced features space representations, one with HDP labels and the other with labels predicted by our DEC model. The clusters' separations increases at each consecutive clustering step and the clouds are denser and denser as the clustering proceeds. An example of this is shown in Fig. 5.10. We can



(a) (b)

Figure 5.7: This figure shows a selection of 4s (a) and 9s (b) for a 2-D reduced features space. The data points are represented by their original image to highlight the differences between the two clouds representing the two different clusters identified by our DEC model.

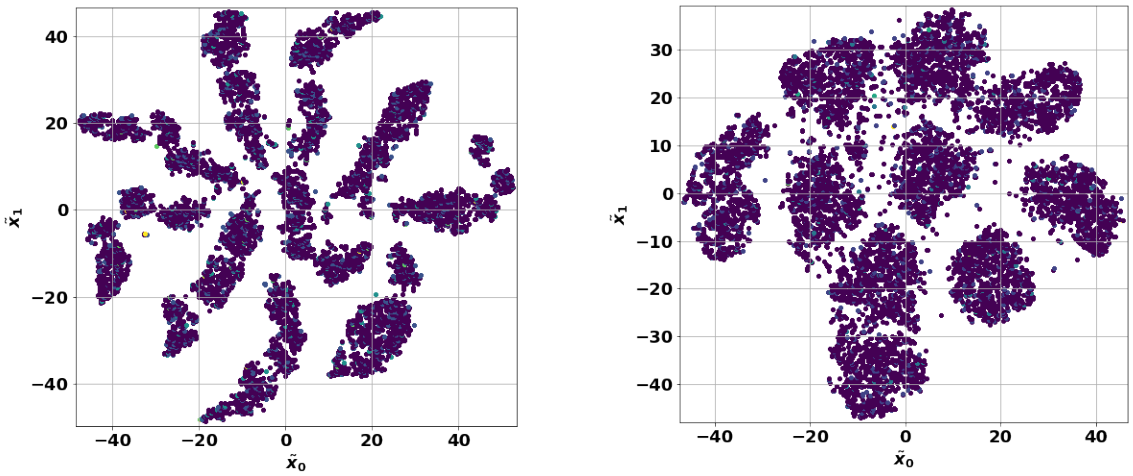
assume that this is a general behaviour of the model since it manifests with different data sets.

The comparison of the metrics chosen for evaluating the clustering w.r.t. HDP labels shows the quality of the clustering of our DEC model and also the amount of information that our clustering carries about HDP clustering. The values reported in Tab. B.6 and, especially, in Tab. B.7 must be therefore compared with the metrics computed using HDP labels, reported in Tab. 5.1. The values obtained using our DEC model are higher for every metric. Metrics computed using HDP labels on feature space do not have any meaning, even though TSAE is able to retain some of the original data structure. On the other hand these metrics computed using HDP labels on the data space are meaningful, as HDP method is performed directly on the data space.

Table 5.1: This table reports the values of the metrics chosen to evaluate the clustering for HDP label assignments.

$Silh_{data}$	$Silh_{feat}$	$\log(CH_{data})$	$\log(CH_{feat})$
0.15051924	-0.14816532	1.891993306	1.76427582

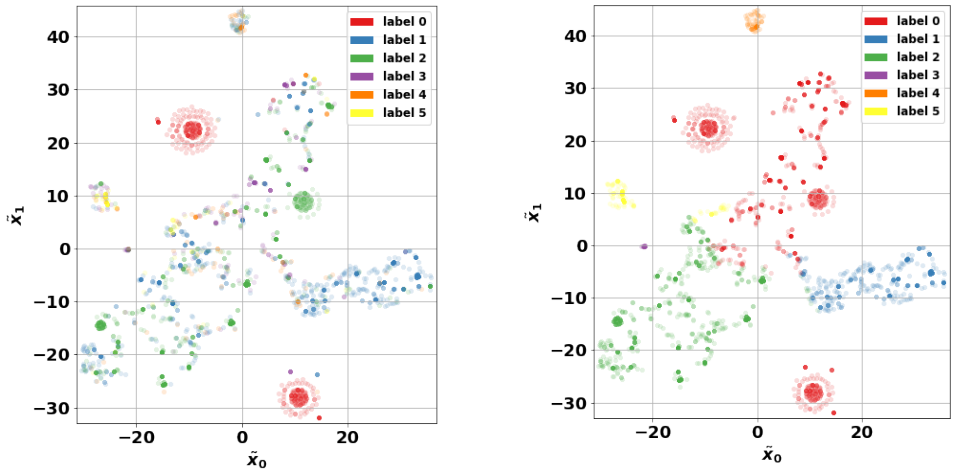
We have made another comparison w.r.t. HDP clustering in terms of the outcomes of patients, those are described in 3.2. Three different events have been collected: leukemia



(a) (b)

Figure 5.8: This figure represents the 2-D reduction, performed using t-SNE, of the feature space for the best performing configuration of our DEC model. On the left (a) at the end of the clustering step, while on the right (b) at the end of the TSAE training. Colors are given as a function of the density, brighter colors mean denser region.

free survival, overall survival, acute myeloid leukemia (AML) adjusted overall survival. For each event, the time at which it appeared (if it appeared) is given along with a integer value that describes if that event happened (when equal to 1) or not (when equal to 0). This is the common description that is used in this context in order to fit the Kaplan–Meier estimator [40], also known as the product limit estimator. This is a non-parametric statistic used to estimate the survival function from lifetime data. In medical research, it is often used to measure the fraction of patients living for a certain amount of time after treatment. The comparisons of these curves between two different label assignments, HDP and our best performing configuration, are shown in Fig. 5.11, 5.12, 5.13. Our labels have been synchronized with HDP labels by solving the "linear sum assignment" problem [41] between the two assignments. Using these curves we are able to perform a comparison between our assignments and HDP assignments on another level of information: the survival probability. These outcomes are the slice of the data set that was never fed to our model, i.e. never-seen data. Different classes produced well separated curves, meaning that our assignments based on the mutations were able to describe different survival curves. Having different survival curves for different clusters means that patients belonging to different clusters will be the patients with different outcomes. Studying the patients belonging to a cluster it is possible to characterize the cluster by identifying the most frequent mutations among these patients. Characterizing the clusters using their most frequent mutations is a common procedure that allows to



(a) (b)

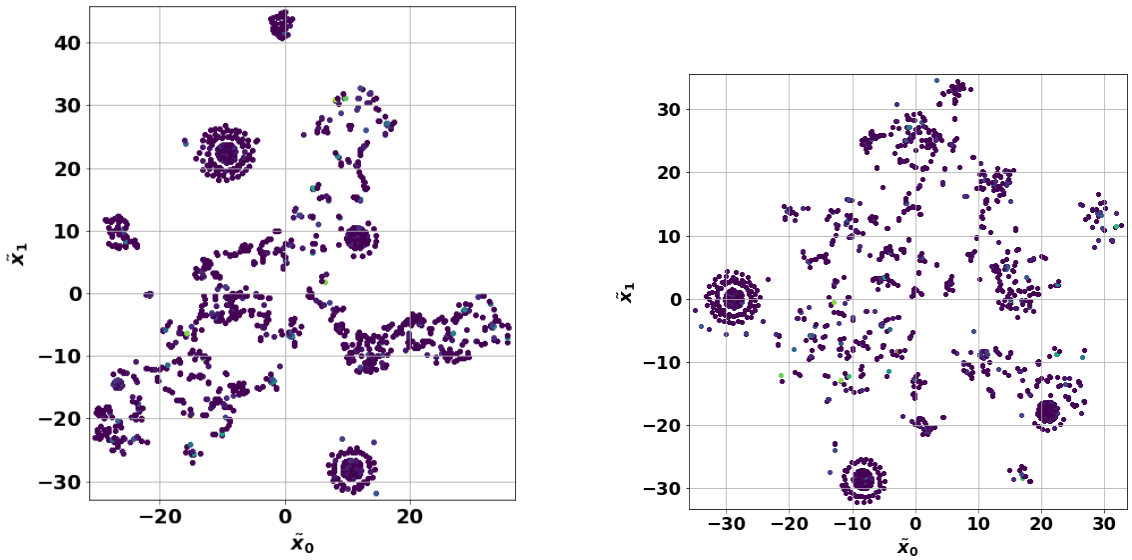
Figure 5.9: This figure represents the 2-D reduction, performed using t-SNE, of the feature space at the end of the clustering step for the best performing configuration of our DEC model. On the left (a) points are labelled by HDP labels, while on the right (b) by our predictions.

promote these mutation as "drivers" of the cluster, i.e. a sort of fingerprint that clinicians could exploit in prognosis and/or diagnosis.

5.5 Generative capabilities of our DEC model

We chose to evaluate the generative capabilities of our DEC model studying the final values of the *Cycle Accuracy* and the *Reconstruction Loss*. We did this study on the results of MNIST and BMNIST data set. In Appendix B, from Tab. B.4 we extracted the plot of *Cycle Accuracy* versus the *Reconstruction Loss* shown in Fig. 5.14. Interpreting the *Cycle Accuracy* as the efficiency of our DEC model to consistently classify its synthetic generations, we can say that it is not possible to have an optimal reconstruction without giving up the efficiency of the model in classifying synthetic data. Examples of reconstructed images for low and high values of reconstruction loss are shown in Fig. 5.15.

The relation between *Cycle Accuracy* and *Reconstruction Loss*, suggests also that the clustering step corrupts the reconstruction of images in a limited way, in fact our DEC model seemed to decide which features are important and what value they must have for representing that cluster. Looking at Fig. 5.16, the final reconstructed images of the elements of a specific cluster are almost equal to each other, meaning that the model has



(a) (b)

Figure 5.10: This figure represents the 2-D reduction, performed using t-SNE, of the feature space for the best performing configuration of our DEC model. On the left (a) at the end of the clustering step, while on the right (b) at the end of the TSAE training. Colors are given as a function of the density, brighter colors mean denser region.

assigned the same specific values to those features, characteristic of that specific cluster.

5.6 Federated implementation results

The following results are meant to highlight that the federated implementation of our DEC model is consistent with the centralized one. Our results are limited to the configurations we were able to test. Federated optimizers were chosen depending on the local optimizer, s.t. FedAvg was coupled with local SGD, FedAdam with local Adam, and FedYogi with local Yogi. The local optimizers used had the same hyperparameters of the best configurations tuned previously. The specific server hyperparameters of FedYogi and FedAdam were set to $\eta_s = 0.01$, $\beta_{1s} = 0.9$ and $\beta_{2s} = 0.999$. At each step, clients performed one local epoch for each federated epoch. We decided to stop federated training of TSAE after 500 epochs, since usually the number of federated epochs must be higher than the centralized counterpart since convergence rates are lower. The aggregation of the centroids has been extensively described in 2.1.2. We stopped federated training of the clustering step, i.e. the minimization of the KLD, after 25 epochs. In the following, Fig. 5.17 some examples of the feature space built by the best federated implementation

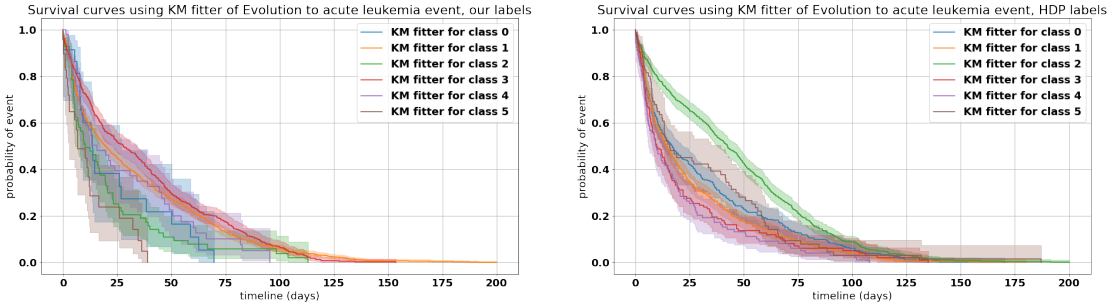


Figure 5.11: Survival curves produced using Kaplan-Meier estimator for the event of leukemia free survival. On the left are reported classes from our labels, on the right from HDP labels.

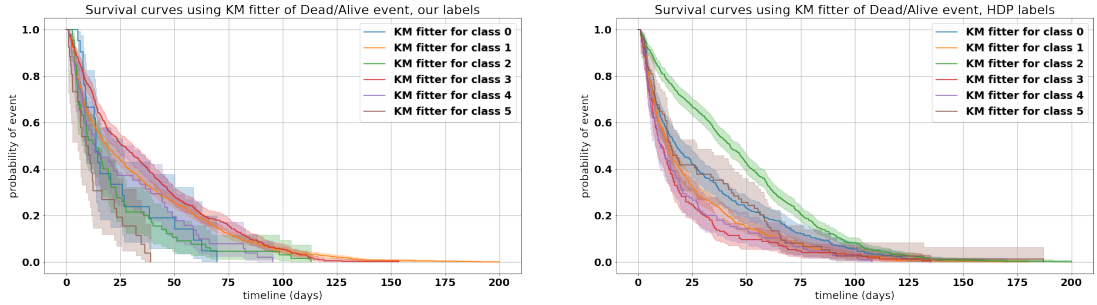


Figure 5.12: Survival curves produced using Kaplan-Meier estimator for the event of overall survival. On the left are reported classes from our labels, on the right from HDP labels.

after training the TSAE and also after the clustering step are shown.

We computed also the metrics obtained by the federated models on the centralized data set. This is usually not allowed in a truly federated setup since evaluation must happen at clients' place on clients' data sets. The central server could in fact obtain only the results of a local evaluation and then it could then perform some sort of aggregation. Sometimes a centralized data set is used for the evaluation at server place, but it must not contain data from clients. Our objective at this stage was to evaluate the consistency of the federated implementation w.r.t. the centralized one, and then to treat the final federated model as a centralized trained model. In Tab. 5.2 and Tab. 5.3 are shown the five best configurations, the others are available in Appendix C, Tab. C.1 and Tab. C.2.

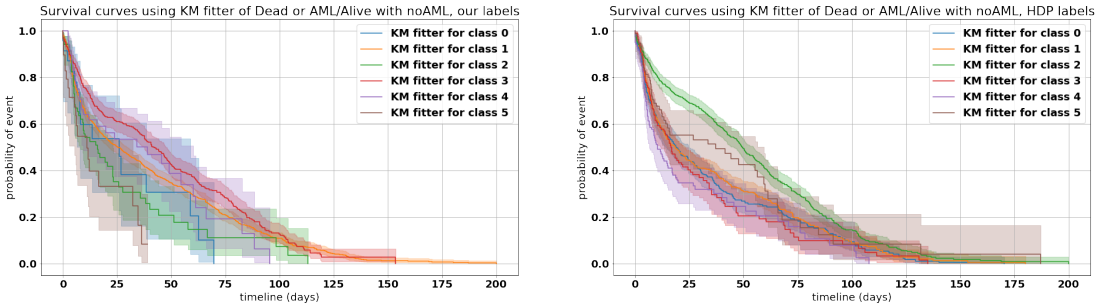
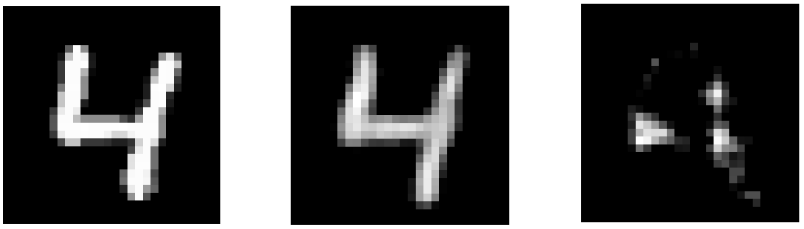


Figure 5.13: Survival curves produced using Kaplan-Meier estimator for the event of AML overall survival. On the left are reported classes from our labels, on the right from HDP labels.

Table 5.2: This table reports the metrics of the five best configurations of the federated implementation of our DEC model between those tested. The first three columns represent the federated configuration, while the others the final values of the metrics studied. They are ordered as explained in 5.2.

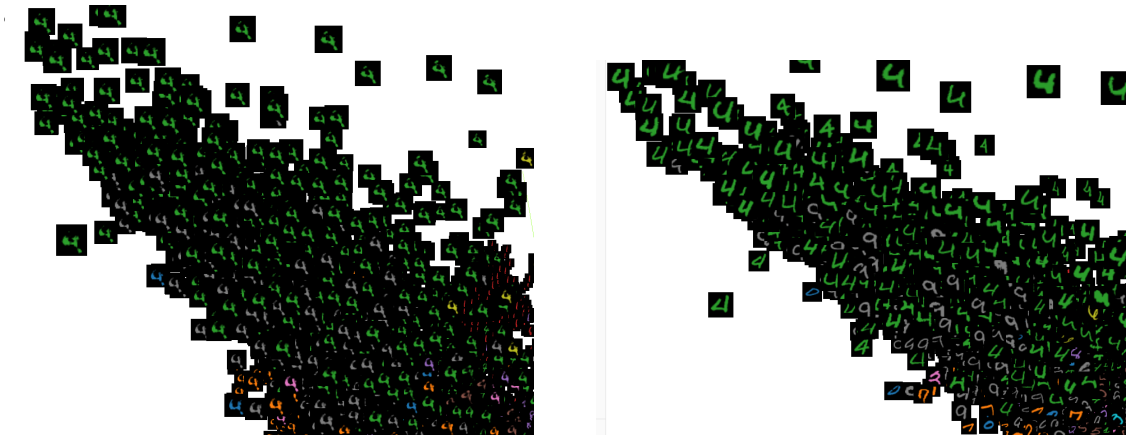
Samples distr.	Optimizers	Centroids agg.	Recon. Loss	Accuracy	Cycle Accuracy	AE Loss
uniform	FEDYOGI	max min	8.77262	0.56192	0.98434	0.01759
uniform	FEDADAM	random	0.04096	0.57024	0.47088	0.01787
skewed Gaussian	FEDADAM	random	0.04124	0.54185	0.25061	0.0173
skewed Gaussian	FEDYOGI	random	2.61656	0.60842	0.81351	0.01606
skewed Gaussian	FEDYOGI	max min	2.80088	0.55605	0.82917	0.01603

Figure 5.14: This plot shows the results of the best configurations of MNIST training, those in Tab. B.4. Especially the *Cycle Accuracy* is plotted against the logarithm of *Reconstruction Loss*.



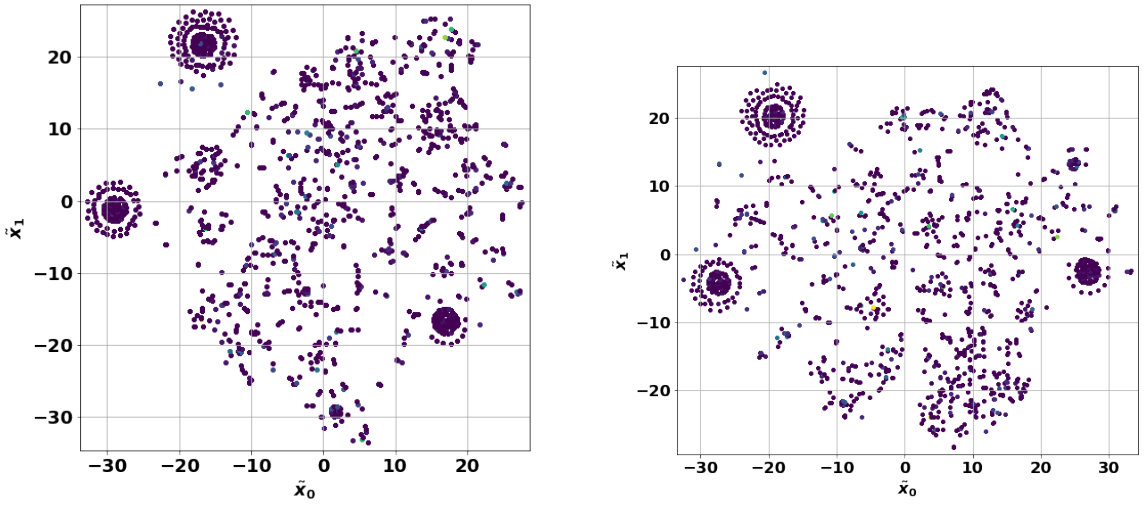
(a) (b) (c)

Figure 5.15: This is a comparison between different reconstructions. On the left (a) the original image of a 4 is shown. In the center (b) the reconstructed image from a trained TSAE with a value of reconstruction loss of 0.014. On the right (c) the reconstructed image from a trained TSAE with a value of reconstruction loss of 0.13.



(a) (b)

Figure 5.16: This figure shows a part of the 2-D reduced features space. On the left (a), data points are represented with their reconstructed representation. On the right (b), data points are represented with their original representation. Data points are colored by actual label. This shows how the clustering step corrupts the reconstructed representation of data points.



(a) (b)

Figure 5.17: This figure represents the 2-D reduction, performed using t-SNE, of the feature space for the best performing configuration of our federated DEC model. On the left (a) at the end of the clustering step, while on the right (b) at the end of the TSAE training. These results come from the federated implementation. Colors are given as a function of the density, brighter colors mean denser region.

Table 5.3: This table reports the metrics of the five best configurations of the federated implementation of our DEC model between those tested. The first three columns represent the federated configuration, while the others the final values of the metrics studied. They are ordered as explained in 5.2.

Samples distr.	**Optimizers**	**Centroids agg.**	$Silh_{data}$	$Silh_{feat}$	$\log(CH_{data})$	$\log(CH_{feat})$
uniform	FEDYOGI	max min	0.14702	0.69714	4.31038	7.93359
uniform	FEDADAM	random	0.14518	0.72065	4.24762	8.57021
skewed Gaussian	FEDADAM	random	0.12026	0.70514	4.14665	8.35007
skewed Gaussian	FEDYOGI	random	0.10614	0.6342	3.80749	7.65483
skewed Gaussian	FEDYOGI	max min	0.10446	0.6232	4.03885	7.67925

Chapter 6

Conclusions

In this thesis work we have explored the unsupervised clustering problem of a very important data set of medical data, i.e. EUROMDS. This data represents the genetic mutations of patients affected by Myelodysplastic Syndrome (MDS), an haematological disease. In this context, clustering of patients is often one of the preliminary procedures before completing diagnosis or prognosis, or even to estimate the survival probability. Since in medical field the data involved is really sensitive and strongly protected under privacy regulations, we developed a machine learning model that could be trained and deployed in a federated learning setting. Federated learning is an innovative distributed machine learning algorithm that is able to achieve the machine learning objective without having to see or touch data. This is a potential breakthrough for medical application of machine learning, because it solves many privacy concerns. Some results of this work have contributed to GenoMed4All European Project productions, especially those referring to applications of federated learning.

We inspired our model on an unsupervised clustering method, called "DEC", that is based on neural networks. Federated learning algorithms are, in the end, methods to aggregate parameters of neural network models trained locally on clients, and that is one of the main reasons that led us to choose "DEC". We decided to modernize "DEC" model exploiting some novel techniques. The modifications we made to the original "DEC" were proposed with the objective to achieve some goals, these are discussed in the following. First, we wanted to simplify the training procedure, especially avoiding a greedy layer-wise pretraining since it is time consuming and not easy to implement in a federated setting. This led us to develop TSAE architecture and to explore regularization techniques. Our architecture could be further improved with a deeper study on the initialization. Furthermore, we wanted to improve the state of the art performance of the original "DEC" model on one of the data sets tested by the authors, i.e. MNIST. We have improved the accuracy by $\sim 6\%$ on MNIST data set, and we have also discovered that this method is affected by systematic errors in discriminating between 4s and 9s. Further improvements could be achieved by the use of convolutional neural network

(CNN) instead of TSAE to parametrize the mapping f_θ. CNNs usage for EUROMDS data should not have any meaning in principle, except if it was to find a semantically meaningful ordering principle for "mutations" in order to build an image representation of a patient, therefore its use was not explored. Third, we wanted to insert in the network architecture, and inside the training procedure also, some data set specific tools. This is motivated by the fact that our data set of interest, i.e. EUROMDS, is composed by binary vectors and not real-valued data. We have tested to this aim a modified mean squared error loss for training the TSAE. Other modifications could be tested in the future to further improve both the reconstruction of the data and the projection to the feature space. The step of TSAE training is fundamental, and exploring the possibility to enable a partial cluster separation in the feature space already at the end of the pretraining step would be a breakthrough. Fourth, we have rethought the complex training procedure in order to enable a federated implementation, in fact this was not possible before. The critical step for enabling a federated implementation was the aggregation of centroids. We developed an iterative procedure of aggregation, and we have also proposed simpler statistical procedures. We tested these obtaining results consistent with the centralized approach. The best aggregation procedure depends heavily on the distribution of the samples, and thus may need further study before being applied to our DEC model to other problems.

We chose two metrics to characterize the quality of our clustering, i.e. Silhouette Score and Calinski Harabasz Score. The final results were compared with the metrics computed using HDP labels, the "gold standard" available for this task. Since HDP method is very different w.r.t. our solution, we compared the metrics on the basis of how much information our clustering carries about HDP clustering. We also plotted the survival curves of the outcome events in the data set, showing that their evolution is quite similar w.r.t. HDP clustering.

We have also explored the generative capabilities of our DEC model. MNIST and BMNIST data sets helped us in this specific study, since they present an image representation of very easy interpretation. We chose two metrics that described the generative capability of our DEC model. We understood that in order to obtain a better reconstruction we have to accept the compromise of losing the cleverness of our DEC model in clustering synthetic data. Moreover, the use of variational autoencoders (VAEs) instead of TSAE could represent a further improvement. VAEs are in fact really promising in reconstructing real data.

Coupling satisfactory generative capabilities with good clustering performance could allow for the model to produce synthetic data that belongs to a specific cluster. For MDS problem, the value representing a synthetic generation should be interpreted as the probability for the mutation to be present. It should also be possible to study the topology of the clusters in feature space and more interestingly in real data space. Using the image representation we presented, it is also possible to provide clinicians with a litmus paper to be used with new patients.

Appendix A

TSDAE hyperparameter tuning

In this section will be reported additional results and graphs from tuning the hyperparameters of TSAE.

Table A.1: This table resumes the best configurations obtained by tuning the learning rate (LR) in training the TSDAE on MNIST. The best configurations are given for every optimizer, with and without the learning rate (LR) scheduler. The final loss resulting is also reported.

LR scheduler	**Optimizer**	**LR**	**Loss**
True	ADAM	0.0007	0.0113
True	SGD	0.6546	0.0138
True	YOGI	0.0644	0.0115
False	ADAM	0.0003	0.0122
False	SGD	0.4784	0.0137
False	YOGI	0.0369	0.0125

Table A.2: This table resumes the best configurations obtained by tuning the learning rate (LR) in training the TSDAE on BMNIST. The best configurations are given for every optimizer, with and without the learning rate (LR) scheduler. The final loss resulting is also reported.

LR scheduler	**Optimizer**	**LR**	**Loss**
True	ADAM	0.0007	0.0227
True	SGD	0.3999	0.0268
True	YOGI	0.0699	0.0232
False	ADAM	0.0002	0.0240
False	SGD	0.7243	0.0244
False	YOGI	0.0241	0.0248

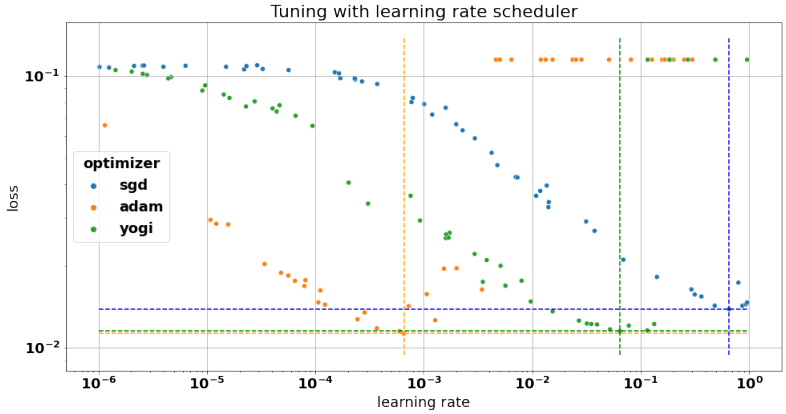


Figure A.1: These are the results for training TSDAE on MNIST data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has been used.

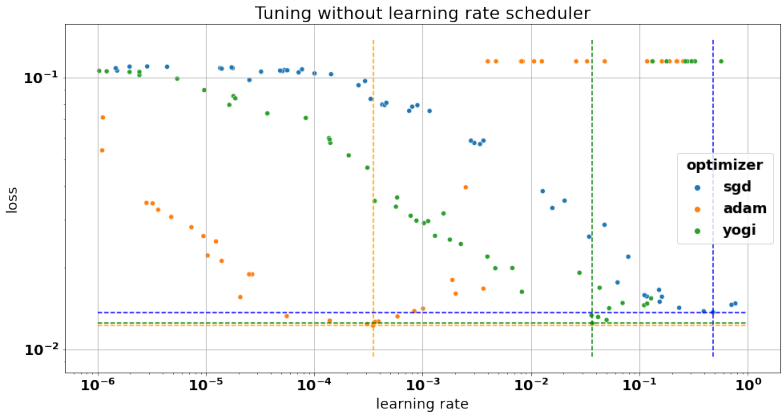


Figure A.2: These are the results for training TSDAE on MNIST data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has not been used.

Table A.3: This table resumes the best configurations obtained by tuning the learning rate (LR) in training the TSDAE on EUROMDS. The best configurations are given for every architecture, for every optimizer, with and without the learning rate (LR) scheduler. The final loss resulting is also reported.

Architecture	**LR scheduler**	**Optimizer**	**LR**	**Loss**
CURVES	False	ADAM	0.0001	0.0086
CURVES	False	SGD	0.0415	0.0103
CURVES	False	YOGI	0.0010	0.0089
CURVES	True	ADAM	0.0002	0.0100
CURVES	True	SGD	0.0373	0.0103
CURVES	True	YOGI	0.0020	0.0103
DEC	False	ADAM	9.5743	0.0035
DEC	False	SGD	0.0036	0.0040
DEC	False	YOGI	0.0009	0.0032
DEC	True	ADAM	9.7019	0.0042
DEC	True	SGD	0.0080	0.0047
DEC	True	YOGI	0.0003	0.0039
GOOGLE	False	ADAM	0.0001	0.0042
GOOGLE	False	SGD	0.0065	0.0043
GOOGLE	False	YOGI	0.0008	0.0042
GOOGLE	True	ADAM	4.0641	0.0043
GOOGLE	True	SGD	0.0140	0.0055
GOOGLE	True	YOGI	0.0014	0.0047

Table A.4: This table resumes the best configurations obtained by tuning activations and batch size in training the TSDAE on EUROMDS. The best configurations are given for every architecture, for every optimizer. The final loss resulting is also reported.

Architecture	**Optimizer**	**Batch Size**	**Hidden Act.**	**Final Act.**	**Loss**
CURVES	ADAM	8	relu	relu	0.0241
CURVES	SGD	8	relu	relu	0.0362
CURVES	YOGI	8	relu	relu	0.0272
DEC	ADAM	16	relu	relu	0.0044
DEC	SGD	16	relu	relu	0.0195
DEC	YOGI	8	relu	relu	0.0082
GOOGLE	ADAM	8	relu	relu	0.0097
GOOGLE	SGD	8	relu	relu	0.0181
GOOGLE	YOGI	8	relu	relu	0.0070

Table A.5: This table resumes the best configurations obtained by tuning different input corruption rates. The best configurations are given for every architecture, for every optimizer. The final loss resulting is also reported.

Architecture	**Corruption Rate**	**Optimizer**	**Loss**
CURVES	0.06688	ADAM	0.0086
CURVES	0.1453	SGD	0.01373
CURVES	0.08083	YOGI	0.00924
DEC	0.0806	ADAM	0.00306
DEC	0.11924	SGD	0.00479
DEC	0.16963	YOGI	0.00311
GOOGLE	0.17797	ADAM	0.00344
GOOGLE	0.07571	SGD	0.00429
GOOGLE	0.08739	YOGI	0.00336

Table A.6: This table resumes the best configurations obtained by tuning different hidden dropout rates. The best configurations are given for every architecture, for every optimizer. The final loss resulting is also reported.

Architecture	**Dropout Rate**	**Optimizer**	**Loss**
CURVES	0.08032	ADAM	0.01305
CURVES	0.02239	SGD	0.01434
CURVES	0.00878	YOGI	0.00973
DEC	0.12112	ADAM	0.00344
DEC	0.03004	SGD	0.00904
DEC	0.04302	YOGI	0.00353
GOOGLE	0.09342	ADAM	0.00317
GOOGLE	0.02184	SGD	0.00586
GOOGLE	0.13257	YOGI	0.00391

Table A.7: This table resumes the best configurations obtained by tuning different values of σ for the input Gaussian noise. The best configurations are given for every architecture, for every optimizer. The final loss resulting is also reported.

Architecture	σ	**Optimizer**	**Loss**
CURVES	0.14674	ADAM	0.00725
CURVES	0.04856	SGD	0.00882
CURVES	0.15848	YOGI	0.00788
DEC	0.18134	ADAM	0.00274
DEC	0.08128	SGD	0.003
DEC	0.17921	YOGI	0.00309
GOOGLE	0.19483	ADAM	0.00255
GOOGLE	0.09676	SGD	0.00448
GOOGLE	0.19978	YOGI	0.00279

Table A.8: This table resumes the best configurations obtained by tuning different values of β for the modified loss. The best configurations are given for every architecture, for every optimizer. The final loss resulting is also reported.

Architecture	β	**Optimizer**	**Loss**
CURVES	0.101	ADAM	0.01231
CURVES	0.02004	SGD	0.01386
CURVES	0.17995	YOGI	0.01191
DEC	0.1809	ADAM	0.00501
DEC	0.04614	SGD	0.00546
DEC	0.0161	YOGI	0.00398
GOOGLE	0.17598	ADAM	0.00589
GOOGLE	0.0159	SGD	0.00578
GOOGLE	0.00309	YOGI	0.00478

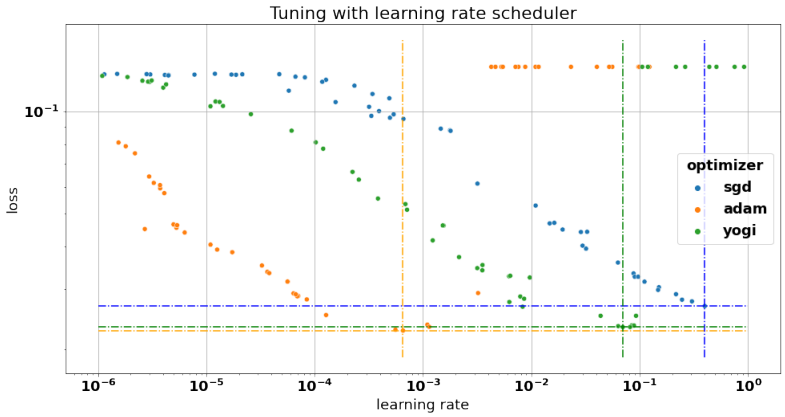


Figure A.3: These are the results for training TSDAE on BMNIST data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has been used.

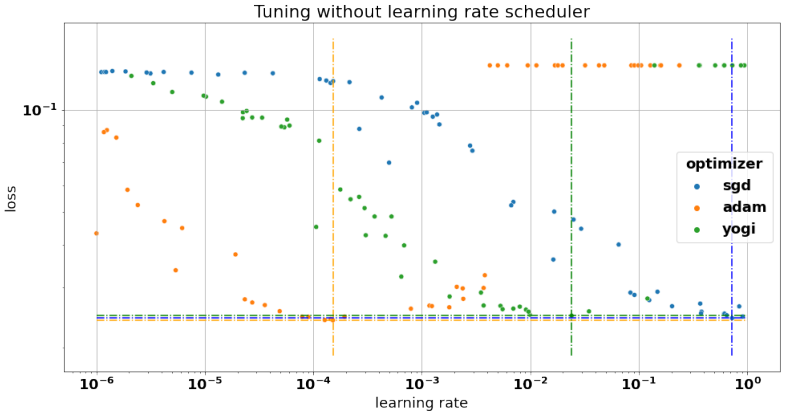


Figure A.4: These are the results for training TSDAE on BMNIST data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has not been used.

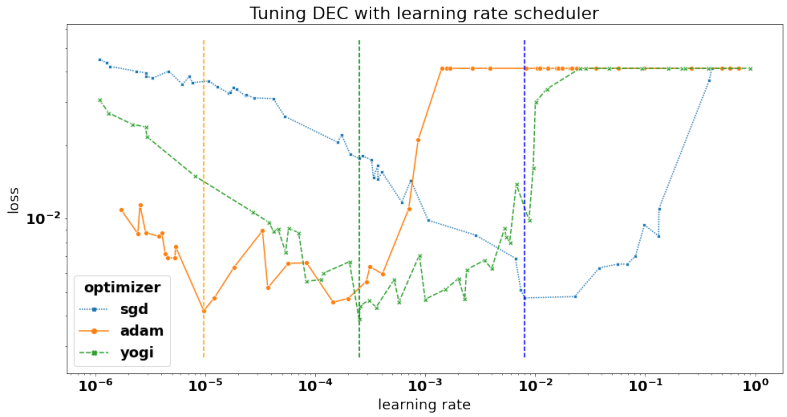


Figure A.5: These are the results for training TSDAE, in "DEC" configuration on EUROMDS data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has been used.

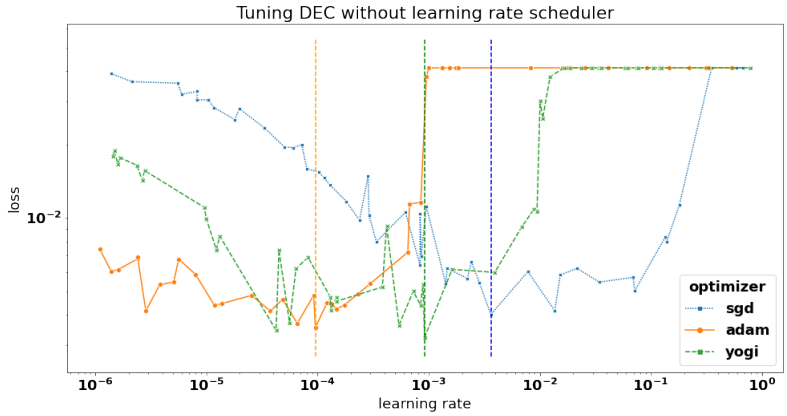


Figure A.6: These are the results for training TSDAE, in "DEC" configuration on EUROMDS data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has not been used.

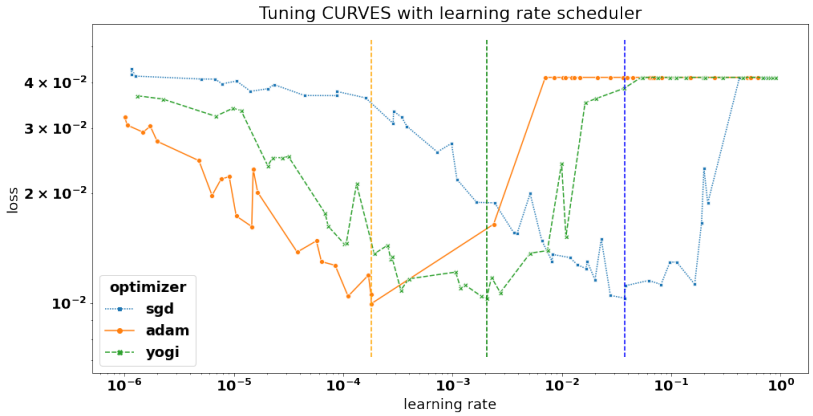


Figure A.7: These are the results for training TSDAE, in "CURVES" configuration on EUROMDS data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has been used.

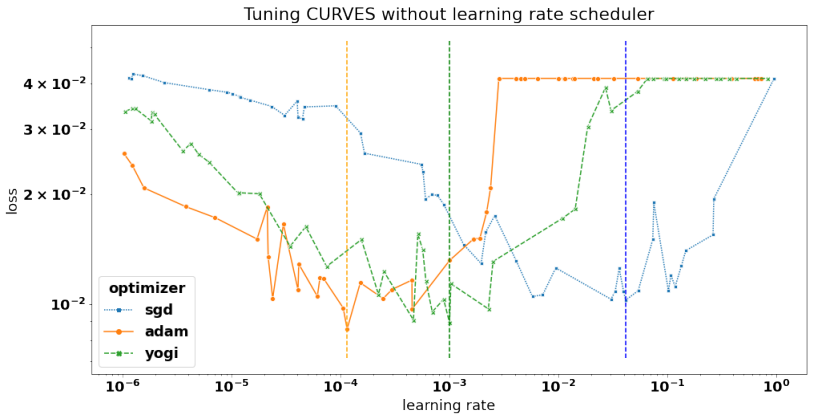


Figure A.8: These are the results for training TSDAE, in "CURVES" configuration on EUROMDS data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has not been used.

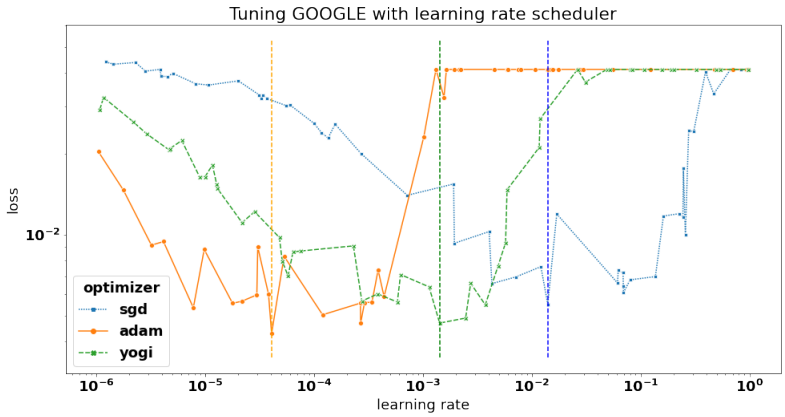


Figure A.9: These are the results for training TSDAE, in "GOOGLE" configuration on EUROMDS data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has been used.

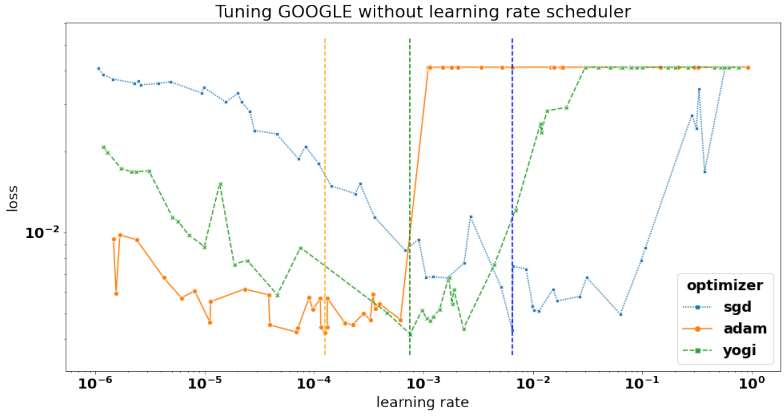


Figure A.10: These are the results for training TSDAE, in "GOOGLE" configuration on EUROMDS data set with the three optimizers, for different values of the learning rate. The best configurations (minimum loss) for each optimizer are highlighted by colored dashed lines, same colors as the scatter points. For these trainings, a learning rate scheduler has not been used.

Appendix B

Clustering step hyperparameter tuning

In this section are reported the results about tuning the hyper parameters of the clustering step of our model.

B.1 Tuning on MNIST and BMNIST

Table B.1: This table resumes the tuning of learning rate and batch size in the clustering step on MNIST data set. Here were used SGD optimizer. Here the parameter α was set to 1, and there was not used any scaler before computing initial centroids with K-Mmeans algorithm. Only the best four results, in accuracy, are reported. Learning rate is abbreviated with LR.

LR	Batch Size	Accuracy	Cycle Accuracy	Loss	Optimizer
0.0427	64.0	0.86775	0.80772	77.65878	sgd
0.02715	64.0	0.8552	0.89512	69.04132	sgd
0.14485	512.0	0.85385	0.99397	3.42269	sgd
0.21511	512.0	0.84913	0.99397	2.88268	sgd
0.16086	256.0	0.84845	0.79875	6.31574	sgd

B.2 Tuning on EUROMDS

Table B.2: This table resumes the tuning of learning rate and batch size in the clustering step on MNIST data set. Here were used Yogi optimizer. Here the parameter α was set to 1, and there was not used any scaler before computing initial centroids with K-Mmeans algorithm. Only the best four results, in accuracy, are reported. Learning rate is abbreviated with LR.

LR	**Batch Size**	**Accuracy**	**Cycle Accuracy**	**Loss**	**Optimizer**
0.00148	64.0	0.8019	0.84267	13.21794	yogi
0.00408	256.0	0.80123	0.8985	3.19888	yogi
0.00155	64.0	0.80015	0.80137	13.05863	yogi
0.00116	64.0	0.79957	0.8577	14.49286	yogi
0.00241	128.0	0.7991	0.94188	6.44391	yogi

Table B.3: This table resumes the tuning of learning rate and batch size in the clustering step on MNIST data set. Here were used Adam optimizer. Here the parameter α was set to 1, and there was not used any scaler before computing initial centroids with K-Mmeans algorithm. Only the best four results, in accuracy, are reported. Learning rate is abbreviated with LR.

LR	**Batch Size**	**Accuracy**	**Cycle Accuracy**	**Loss**	**Optimizer**
0.00102	256.0	0.85807	0.60787	2.80548	adam
0.00059	128.0	0.85517	0.56227	6.02143	adam
0.00127	512.0	0.85352	0.5123	1.31773	adam
0.00038	128.0	0.85298	0.612	6.38374	adam
0.0003	64.0	0.85085	0.50397	12.97376	adam

Table B.4: This table resumes all the results of training our DEC model on MNIST using the best values for the learning rate and the batch size for each optimizer. During these 24 trainings were used the four different scaler, the two different values of α that we chose to test. Δ_{label} describes the percentage of variation of labels of the last iteration w.r.t. to the previous. The results are reported ordered in decreasing order by accuracy.

Alpha	**Scaler**	**Accuracy**	**Cycle Accuracy**	**Loss**	Δ_{label}	**Optimizer**
9.0	normal-l1	0.88235	0.98682	14.0735	0.04472	yogi
9.0	none	0.87752	0.92863	15.26898	0.05312	sgd
9.0	normal-l2	0.87182	0.99327	15.41405	0.03305	yogi
9.0	normal-l1	0.86617	0.98727	15.52427	0.07432	sgd
1.0	normal-l2	0.8628	0.58535	2.84652	0.01253	adam
9.0	normal-l2	0.86167	0.98177	17.89363	0.08538	sgd
9.0	standard	0.85697	0.8771	13.07404	0.07412	sgd
9.0	standard	0.85313	0.90305	15.67815	0.04265	yogi
1.0	normal-l1	0.8499	0.64462	12.90239	0.01745	yogi
1.0	normal-l2	0.84943	0.79785	83.94711	0.03705	sgd
1.0	normal-l1	0.84868	0.88817	88.14906	0.03962	sgd
1.0	normal-l1	0.84798	0.3722	2.8952	0.01543	adam
1.0	standard	0.8465	0.50638	2.95308	0.0203	adam
1.0	normal-l2	0.84592	0.81267	13.37499	0.0054	yogi
1.0	none	0.84448	0.70637	60.88186	0.046	sgd
1.0	none	0.8427	0.50768	3.29932	0.03087	adam
9.0	none	0.83842	0.49775	2.70315	0.06738	adam
9.0	normal-l1	0.83508	0.32487	2.73572	0.12912	adam
9.0	normal-l2	0.83287	0.40542	2.67816	0.05003	adam
9.0	standard	0.83203	0.39427	2.67644	0.05823	adam
9.0	none	0.8225	0.99455	17.12021	0.0653	yogi
1.0	standard	0.82223	0.89523	67.49209	0.07438	sgd
1.0	none	0.80033	0.84643	13.79606	0.03992	yogi
1.0	standard	0.79913	0.835	15.08306	0.02552	yogi

Table B.5: This table resumes all the results of training our DEC model on BMNIST using the best values for the learning rate and the batch size for each optimizer. During these 24 trainings were used the four different scaler, the two different values of α that we chose to test. Δ_{label} describes the percentage of variation of labels of the last iteration w.r.t. to the previous. The results are reported ordered in decreasing order by accuracy.

Alpha	**Scaler**	**Accuracy**	**Cycle Accuracy**	**Loss**	Δ_{label}	**Optimizer**
9.0	normal-l2	0.89318	0.99375	17.77081	0.02987	yogi
9.0	normal-l1	0.88955	0.9317	15.72735	0.0395	yogi
9.0	none	0.88475	0.9109	19.67695	0.0894	sgd
9.0	normal-l1	0.86998	0.991	18.60929	0.04082	sgd
9.0	standard	0.86592	0.95553	17.28764	0.08655	sgd
9.0	none	0.8651	0.98983	17.85925	0.03165	yogi
1.0	normal-l2	0.8501	0.7277	22.37781	0.01273	sgd
9.0	normal-l2	0.84518	0.9935	17.83402	0.0794	sgd
1.0	normal-l2	0.84272	0.80915	13.82713	0.00323	yogi
1.0	normal-l1	0.83477	0.6138	72.59322	0.05692	sgd
9.0	standard	0.8323	0.98933	24.66313	0.09432	yogi
1.0	normal-l1	0.83032	0.7308	14.61929	0.00248	yogi
1.0	none	0.79293	0.74127	39.69674	0.03415	sgd
9.0	none	0.79088	0.29445	3.53923	0.09833	adam
1.0	standard	0.7902	0.89635	48.61731	0.0445	sgd
9.0	normal-l2	0.78245	0.4991	3.80168	0.10355	adam
1.0	normal-l1	0.76648	0.5639	4.3915	0.0377	adam
9.0	normal-l1	0.75987	0.49943	3.70059	0.11902	adam
9.0	standard	0.74818	0.31305	3.33674	0.10095	adam
1.0	normal-l2	0.72077	0.4259	3.99873	0.03725	adam
1.0	standard	0.71898	0.6246	3.91656	0.0372	adam
1.0	none	0.71257	0.89955	13.73243	0.00903	yogi
1.0	none	0.70258	0.47568	3.83215	0.04055	adam
1.0	standard	0.69265	0.77763	14.5494	0.02408	yogi

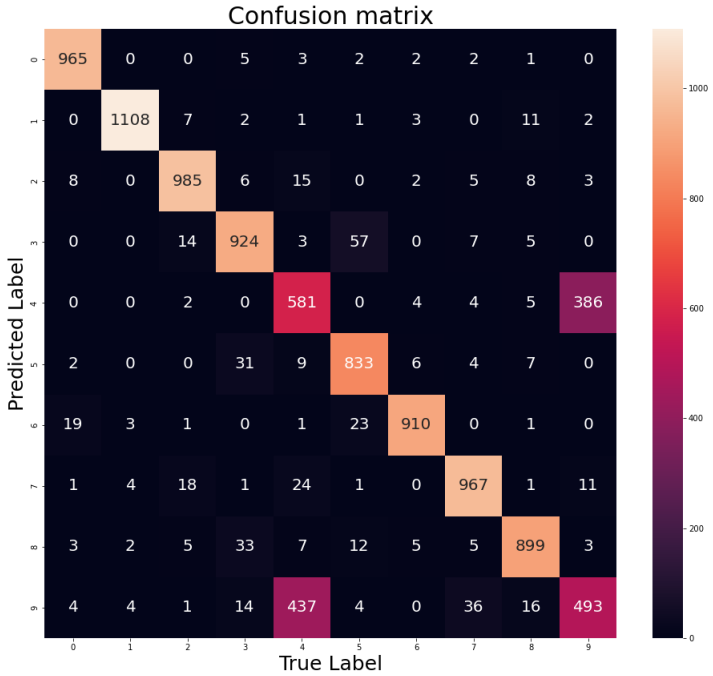


Figure B.1: Confusion matrix for the accuracy of the best configuration of our DEC model on BMNIST data set.

Table B.6: This table resumes the tuning of learning rate and batch size in the clustering step on EUROMDS data set. Here were used all the three optimizers, and all the three architectures. The results are ordered as explained in 5.2. Here the parameter α was set to 1, and there was not used any scaler before computing initial centroids with K-Mmeans algorithm. Only the best ten results are reported. Here the metrics of importance are accuracy, cycle accuracy, Δ_label. This last describes the percentage of variation of labels of the last iteration w.r.t. to the previous. Batch size (BS), learning rate (LR), architecture, and optimizer are the variables describing the specific configuration.

BS	**LR**	**Accuracy**	**Cycle Accuracy**	**Loss**	Δ_{label}	**Architecture**	**Optimizer**
64.0	0.003	0.61527	0.99853	0.46623	0.00098	dec	yogi
16.0	3e-05	0.57954	0.7812	0.03438	0.00049	google	yogi
64.0	0.00171	0.58101	0.99706	0.1005	0.00049	google	yogi
64.0	0.33354	0.58737	1.0	10.58277	0.00049	google	sgd
32.0	0.03523	0.56486	0.83994	0.54556	0.00098	google	sgd
8.0	6e-05	0.60499	0.98972	0.04661	0.00098	google	yogi
16.0	0.00902	0.51884	1.0	2.10777	0.0	google	yogi
64.0	0.02881	0.558	0.99755	1.39768	0.00049	dec	sgd
64.0	0.15064	0.57954	0.84288	3.64853	0.00098	google	sgd
8.0	0.00338	0.51787	1.0	0.94229	0.00049	google	yogi

Table B.7: This table resumes the tuning of learning rate and batch size in the clustering step on EUROMDS data set. Here were used all the three optimizers, and all the three architectures. The results are ordered as explained in 5.2. Here the parameter α was set to 1, and there was not used any scaler before computing initial centroids with K-Mmeans algorithm. Only the best ten results are reported. Here the metrics of importance are the silhouette scores (in data space, and feature space), the logarithm of Calinski-Harabasz scores (in data space, and feature space). Batch size (BS), learning rate (LR), architecture, and optimizer are the variables describing the specific configuration.

BS	**LR**	$Silh_{data}$	$Silh_{feat}$	$\log(CH_{data})$	$\log(CH_{feat})$	**Architecture**	**Optimizer**
64.0	0.003	0.23511	0.90038	1.5872	2.33196	dec	yogi
16.0	3e-05	0.2292	0.60418	1.58101	2.044	google	yogi
64.0	0.00171	0.22744	0.87475	1.57643	2.28839	google	yogi
64.0	0.33354	0.22653	0.93831	1.58952	2.40162	google	sgd
32.0	0.03523	0.22635	0.91555	1.59677	2.36284	google	sgd
8.0	6e-05	0.22614	0.79541	1.57575	2.17713	google	yogi
16.0	0.00902	0.22578	0.95155	1.60622	2.41199	google	yogi
64.0	0.02881	0.22552	0.89419	1.58881	2.3045	dec	sgd
64.0	0.15064	0.22385	0.94217	1.59586	2.42799	google	sgd
8.0	0.00338	0.22103	0.95283	1.59626	2.44302	google	yogi

Appendix C

Federated Implementation

Here are shown the final results about the different configurations we tested in the federated implementation of our DEC model.

Table C.1: This table reports the metrics of all the configurations of the federated implementation of our DEC model between those tested. The first three columns represent the federated configuration, while the other the final values of the metrics studied. They are ordered as explained in 5.2.

Samples distr.	Optimizers	Centroids agg.	Recon. Loss	Accuracy	Cycle Accuracy	AE Loss
uniform	FEDYOGI	max min	8.77262	0.56192	0.98434	0.01759
uniform	FEDADAM	random	0.04096	0.57024	0.47088	0.01787
skewed Gaussian	FEDADAM	random	0.04124	0.54185	0.25061	0.0173
skewed Gaussian	FEDYOGI	random	2.61656	0.60842	0.81351	0.01606
skewed Gaussian	FEDYOGI	max min	2.80088	0.55605	0.82917	0.01603
skewed Gaussian	FEDAVG	random weighted	0.57802	0.50367	0.80715	0.01911
skewed Gaussian	FEDAVG	max min	0.75641	0.51787	0.5766	0.01909
uniform	FEDYOGI	double kmeans	2.26761	0.50024	0.8977	0.01754
uniform	FEDYOGI	random	4.2278	0.40088	0.77582	0.01748
uniform	FEDAVG	random	0.38254	0.48507	0.96916	0.02218
skewed Gaussian	FEDYOGI	double kmeans	2.2444	0.41801	0.77239	0.01601
skewed Gaussian	FEDAVG	double kmeans	0.18046	0.55605	0.84043	0.01903
skewed Gaussian	FEDYOGI	random weighted	1.01715	0.43465	0.744	0.0159
skewed Gaussian	FEDADAM	double kmeans	0.04124	0.52227	0.29515	0.01723
uniform	FEDAVG	random weighted	0.14701	0.48654	0.98825	0.02226
skewed Gaussian	FEDADAM	max min	0.42645	0.52178	0.39892	0.01742
uniform	FEDADAM	max min	0.04169	0.4836	0.49241	0.01793
uniform	FEDADAM	random weighted	0.04189	0.48948	0.38326	0.01786
skewed Gaussian	FEDADAM	random weighted	0.04123	0.54136	0.37445	0.01731
skewed Gaussian	FEDAVG	random	0.22358	0.4699	0.83554	0.01935
uniform	FEDAVG	max min	0.07096	0.50857	0.71072	0.02207
uniform	FEDADAM	double kmeans	0.04124	0.44787	0.34508	0.01786
uniform	FEDYOGI	random weighted	1.18649	0.48458	0.56975	0.01763
uniform	FEDAVG	double kmeans	0.05656	0.55702	0.88644	0.02213

Table C.2: This table reports the metrics of all the configurations of the federated implementation of our DEC model between those tested. The first three columns represent the federated configuration, while the other the final values of the metrics studied. They are ordered as explained in 5.2.

Samples distr.	**Optimizers**	**Centroids agg.**	$Silh_{data}$	$Silh_{feat}$	$\log(CH_{data})$	$\log(CH_{feat})$
uniform	FEDYOGI	max min	0.14702	0.69714	4.31038	7.93359
uniform	FEDADAM	random	0.14518	0.72065	4.24762	8.57021
skewed Gaussian	FEDADAM	random	0.12026	0.70514	4.14665	8.35007
skewed Gaussian	FEDYOGI	random	0.10614	0.6342	3.80749	7.65483
skewed Gaussian	FEDYOGI	max min	0.10446	0.6232	4.03885	7.67925
skewed Gaussian	FEDAVG	random weighted	0.09398	0.27914	3.40067	6.12165
skewed Gaussian	FEDAVG	max min	0.08585	0.27174	3.96052	6.19145
uniform	FEDYOGI	double kmeans	0.08392	0.64788	3.89911	7.9153
uniform	FEDYOGI	random	0.08195	0.55724	4.00766	7.50386
uniform	FEDAVG	random	0.08054	0.14137	3.09767	6.01782
skewed Gaussian	FEDYOGI	double kmeans	0.07746	0.60406	3.93928	7.85986
skewed Gaussian	FEDAVG	double kmeans	0.07714	0.19629	3.49227	5.82638
skewed Gaussian	FEDYOGI	random weighted	0.07347	0.7072	3.90681	8.6849
skewed Gaussian	FEDADAM	double kmeans	0.07332	0.64387	3.93511	8.37466
uniform	FEDAVG	random weighted	0.06921	0.20613	3.22043	6.14329
skewed Gaussian	FEDADAM	max min	0.06771	0.61239	3.76844	7.86239
uniform	FEDADAM	max min	0.06752	0.6755	3.65534	8.00904
uniform	FEDADAM	random weighted	0.0521	0.51379	3.62878	7.70888
skewed Gaussian	FEDADAM	random weighted	0.04798	0.652	3.53364	8.37406
skewed Gaussian	FEDAVG	random	0.04591	0.19776	3.65143	5.81864
uniform	FEDAVG	max min	0.04477	0.17052	3.7253	5.49092
uniform	FEDADAM	double kmeans	0.03643	0.69534	3.8747	8.56888
uniform	FEDYOGI	random weighted	0.02751	0.35708	3.56198	8.24578
uniform	FEDAVG	double kmeans	0.02604	0.1559	3.59329	5.25829

Bibliography

[1] E. Parliament, "General Data Protection Regulation (GDPR)," 2018. [Online]. Available: https://gdpr-info.eu/

[2] K. El Emam, F. K. Dankar, R. Issa, E. Jonker, D. Amyot, E. Cogo, J.-P. Corriveau, M. Walker, S. Chowdhury, R. Vaillancourt, T. Roffey, and J. Bottomley, "A globally optimal k-anonymity method for the de-identification of health data," *Journal of the American Medical Informatics Association : JAMIA*, vol. 16, no. 5, pp. 670–682, 2009, 19567795[pmid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/19567795

[3] R. K. Taira, A. A. T. Bui, and H. Kangarloo, "Identification of patient name references within medical documents using semantic selectional restrictions," *Proceedings. AMIA Symposium*, pp. 757–761, 2002, 12463926[pmid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/12463926

[4] S. M. Thomas, B. W. Mamlin, G. Schadow, and C. J. McDonald, "A successful technique for removing names in pathology reports using an augmented search and replace method," *Proceedings. AMIA Symposium*, pp. 777–81, 2002.

[5] U. D. of Health & Human Services. (1996) Health information privacy. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/privacy/special-topics/de-identification/index.html

[6] Institute of Medicine (US) Committee on Health Research and the Privacy of Health Information: The HIPAA Privacy Rule, *Beyond the HIPAA Privacy Rule: Enhancing Privacy, Improving Health Through Research*. Washington (DC): National Academies Press (US), 2009.

[7] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, Sep. 1999. [Online]. Available: https://doi.org/10.1145/331499.331504

[8] R. Sokal, C. Michener, and U. of Kansas, *A Statistical Method for Evaluating Systematic Relationships*, ser. University of Kansas science bulletin.

University of Kansas, 1958. [Online]. Available: https://books.google.it/books?id=o1BlHAAACAAJ

[9] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, "Clustergan : Latent space clustering in generative adversarial networks," *CoRR*, vol. abs/1809.03627, 2018. [Online]. Available: http://arxiv.org/abs/1809.03627

[10] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," *CoRR*, vol. abs/1511.06335, 2015.

[11] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

[12] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2020.

[13] E. Papaemmanuil, M. Gerstung, L. Bullinger, V. I. Gaidzik, P. Paschka, N. D. Roberts, N. E. Potter, M. Heuser, F. Thol, N. Bolli *et al.*, "Genomic classification and prognosis in acute myeloid leukemia," *New England Journal of Medicine*, vol. 374, no. 23, pp. 2209–2221, 2016.

[14] M. Bersanelli, E. Travaglino, M. Meggendorfer, T. Matteuzzi, C. Sala, E. Mosca, C. Chiereghin, N. Di Nanni, M. Gnocchi, M. Zampini *et al.*, "Classification and personalized prognostic assessment on the basis of clinical and genomic features in myelodysplastic syndromes," *J Clin Oncol.*, 2021.

[15] T. Matsutani and M. Hamada, "Parallelized latent dirichlet allocation provides a novel interpretability of mutation signatures in cancer genomes," *Genes*, vol. 11, no. 10, 2020. [Online]. Available: https://www.mdpi.com/2073-4425/11/10/1127

[16] M. Sheller, B. Edwards, G. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. Colen, and S. Bakas, "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Scientific Reports*, vol. 10, 07 2020.

[17] J. Xu and F. Wang, "Federated learning for healthcare informatics," *CoRR*, vol. abs/1911.06270, 2019. [Online]. Available: http://arxiv.org/abs/1911.06270

[18] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. A. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova,

F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," *CoRR*, vol. abs/1912.04977, 2019. [Online]. Available: http://arxiv.org/abs/1912.04977

[19] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2017.

[20] W. de Brouwer. (2019) The federated future is ready for shipping. [Online]. Available: https://doc.ai/blog/federated-future-ready-shipping

[21] EU CORDIS. (2019) Machine learning ledger orchestration for drug discovery. [Online]. Available: https://cordis.europa.eu/project/rcn/223634/factsheet/en?WT.mc_id=RSS-Feed&WT.rss_f=project&WT.rss_a=223634&WT.rss_ev=a

[22] FeatureCloud. (2019) Featurecloud: Our vision. [Online]. Available: https://featurecloud.eu/about/our-vision/

[23] ai.intel. (2019) Federated learning for medical imaging. [Online]. Available: https://www.intel.ai/federated-learning-for-medical-imaging/

[24] P. Courtiol, C. Maussion, M. Moarii, E. Pronier, S. Pilcer, M. Sefta, P. Manceron, S. Toldo, M. Zaslavskiy, N. Le Stang, N. Girard, O. Elemento, A. G. Nicholson, J.-Y. Blay, F. Galateau-Sallé, G. Wainrib, and T. Clozel, "Deep learning-based classification of mesothelioma improves prediction of patient outcome," *Nat Med*, vol. 25, no. 10, pp. 1519–1525, Oct. 2019.

[25] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=LkFG3lB13U5

[26] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," 2021.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International*

Conference on Artificial Intelligence and Statistics, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[30] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/90365351ccc7437a1309dc64e4db32a3-Paper.pdf

[31] L. van der Maaten, "Learning a parametric embedding by preserving local structure," in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, D. van Dyk and M. Welling, Eds., vol. 5. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, 16–18 Apr 2009, pp. 384–391. [Online]. Available: https://proceedings.mlr.press/v5/maaten09a.html

[32] K. Nigam and R. Ghani, "Understanding the behavior of co-training," *KDD-2000 Workshop on Text Mining*, 08 2000.

[33] Yann LeCun and Corinna Cortes and Christopher J.C. Burges. (1998) The mnist database. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[34] GenoMed4ALL Team. (2021) Genomed4all project. [Online]. Available: https://genomed4all.eu/

[35] Cordis, European Commision. (2021) Genomics and personalized medicine for all though artificial intelligence in haematological diseases, genomed4all. [Online]. Available: https://cordis.europa.eu/project/id/101017549

[36] Ray developers. (2022) Ray tune. [Online]. Available: https://docs.ray.io/en/latest/tune/index.html

[37] L. Li, K. Jamieson, A. Rostamizadeh, K. Gonina, M. Hardt, B. Recht, and A. Talwalkar, "Massively parallel hyperparameter tuning," 2018. [Online]. Available: https://openreview.net/forum?id=S1Y7OOlRZ

[38] PyTorch Developers. (2022) Pytorch. [Online]. Available: https://www.pytorch.org/

[39] Lorenzo Sani. (2019) Master thesis project. [Online]. Available: https://www.github.com/relogu/Federated-Learning-Project

[40] E. L. Kaplan and P. Meier, "Nonparametric estimation from incomplete observations," *Journal of the American Statistical Association*, vol. 53, no. 282, pp. 457–481, 1958. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1958.10501452

[41] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109