Universidad Autónoma de Madrid
Escuela politécnica superior

Máster Universitario en Bioinformática y Biología Computacional

Trabajo Fin de Máster

ANÁLISIS DE DATOS DE CITOMETRÍA DE FLUJO MEDIANTE EL USO DE DOMAIN-ADVERSARIAL AUTOENCODERS

ANALYSIS OF FLOW CYTOMETRY DATA WITH DOMAIN-ADVERSARIAL AUTOENCODERS

Autor: Sara Dorado Alfaro
Tutor: Ángela Fernández Pascual
Cotutor: Daniel Jiménez Carretero
Ponente: José Ramón Dorronsoro Ibero

SEPTIEMBRE 2020

ANÁLISIS DE DATOS DE CITOMETRÍA DE FLUJO MEDIANTE EL USO DE DOMAIN-ADVERSARIAL AUTOENCODERS

ANALYSIS OF FLOW CYTOMETRY DATA WITH DOMAIN-ADVERSARIAL AUTOENCODERS

Autor: Sara Dorado Alfaro
Tutor: Ángela Fernández Pascual
Cotutor: Daniel Jiménez Carretero
Ponente: José Ramón Dorronsoro Ibero

Grupo de Aprendizaje Automático (GAA)
Centro Nacional de Investigaciones Cardiovasculares (CNIC)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

SEPTIEMBRE 2020

Resumen

Resumen

El Aprendizaje Automático es un campo de la Inteligencia Artificial enfocado en el análisis automático de datos. En la era del *big data*, aparecen algoritmos que permiten el análisis de cantidades ingentes de datos de forma eficiente, permitiendo incorporar cada vez más conocimiento en nuestros estudios. Uno de los principales campos de aplicación de estos algoritmos es la bioinformática, donde típicamente se analizan grandes cantidades de datos de alta dimensionalidad. No obstante, una de las principales dificultades en el análisis automático de datos de origen biológico es la variación inevitable que se produce en las condiciones experimentales, ocasionando los conocidos efectos de *batch*. Esto dificulta la integración de experimentos que proceden de distintas fuentes experimentales, reduciendo así la capacidad simultánea de análisis y perdiendo información biológica relevante.

Enfocado en experimentos de citometría de flujo, en este trabajo proponemos un nuevo algoritmo con el objetivo de suavizar la influencia de los efectos de batch simultáneamente en un número arbitrario de condiciones experimentales. Aplicando técnicas del estado del arte en Aprendizaje Automático, como adaptación de dominio y aprendizaje adversario, presentamos el *domain-adversarial autoencoder* (DAE). Para la validación del DAE como algoritmo de adaptación de dominio o normalización de *batch*, en este trabajo se realizan experimentos con tres conjuntos de datos. Los dos primeros son datos sencillos y artificiales compuestos de *beads*. En uno de ellos, se simula de forma artificial el atasco o desalineamiento del citómetro. En el otro, tenemos los mismos datos analizados en dos máquinas distintas. El tercer ejemplo es un conjunto de datos real compuesto de células dendítricas de ratón que también han sido recogidos en dos citómetros distintos.

En primer lugar, mostramos cómo estos efectos de batch influyen en los análisis típicamente aplicados por los usuarios de citometría de flujo, como el clustering con *Phenograph* o la visualización con t-SNE. Después, veremos como el DAE consigue eficientemente paliar los efectos de batch en estos ejemplos y mejorar los resultados, consiguiendo un incremento notable del $F1$-score en clustering antes y después de la corrección. Además, se realiza una evaluación visual de las representaciones en espacios de dos dimensiones mediante un autoencoder estándar (SAE), t-SNE y un DAE.

Adicionalmente, en este trabajo se presenta una forma novedosa de evaluar la calidad de la normalización de los datos por batch o experimento mediante distancias estadísticas. En particular, utilizamos la versión multidimensional de la distancia entre distribuciones de Kolmogorov-Smirnov. Veremos como la distribución de los datos en la representación latente del DAE es muy parecida en datos procedentes de diferentes experimentos, presentando una distancia más pequeña que en el caso del SAE, donde no proporcionamos al algoritmo información del dominio para su entrenamiento.

Por lo tanto, este trabajo permite concluir que la adaptación de dominio en datos de citometría de flujo abre una nueva línea de investigación, que proporciona herramientas para la integración de datos procedentes de distintos experimentos.

Palabras Clave

Citometría de flujo, Efectos de batch, Autoencoders, Aprendizaje no supervisado, Adaptación de dominio, Clustering, Reducción de la dimensionalidad.

Abstract

Machine Learning is a field of Artificial Intelligence focused on automatic data analysis. In the era of big data, there appear algorithms that allow the analysis of large quantities of data efficiently, incorporating more knowledge into our studies. One of the main fields of application for these algorithms is bioinformatics, where large amounts of high-dimensional data are typically analyzed. However, one of the main difficulties in the automatic analysis of data with a biological origin is the inevitable variation that occurs in the experimental conditions, causing the well-known batch effects. This makes it difficult to integrate data that come from different experimental sources, thus reducing the simultaneous capacity for analysis and losing relevant biological information.

Focused on flow cytometry data, in this work we propose a new algorithm in the context of unsupervised learning with the aim of smoothing the influence of batch effects simultaneously under an arbitrary number of experimental conditions. Applying state-of-the-art techniques in Machine Learning, such as domain adaptation and adversarial learning, we present the domain-adversarial autoencoder (DAE). For the validation of the DAE as a domain adaptation or batch normalization algorithm, in this work we carry out experiments with three data sets. The first two are simple, artificial datasets composed of *beads* that have been passed through the cytometer in a controlled environment. In one of them, the clogging or misalignment of the cytometer is artificially simulated. In the other, we have the same data analyzed on two different machines. The third example is a real dataset with dendritic cells of mice that have also been collected on two different cytometers.

Firstly, we show how these batch effects influence the analysis typically applied by flow cytometry users, such as clustering with *Phenograph* or visualization with t-SNE. Secondly, we see how the DAE manages to efficiently alleviate the batch effects in these examples and improve the clustering results, achieving a notable increase in the $F1$-score after the correction. In addition, we provide with a visual evaluation of the representations in two-dimensional spaces learnt with a standard autoencoder (SAE), t-SNE and a DAE.

Additionally, in this work we present a novel method to evaluate the quality of the batch normalization of data using statistical distances. In particular, we use the multidimensional version of the Kolmogorov-Smirnov distance between distributions. We show that the distribution of the data in the latent representation of the DAE is very similar when the data comes from different experiments, presenting a smaller distance than in the case of the SAE, where we do not provide the algorithm with domain information in the training step.

Therefore, this work allows us to conclude that domain adaptation in flow cytometry data opens a new line of research, which is focused in developing tools for the integration of data from different experiments.

Key words

Flow cytometry, Batch effects, Autoencoders, Unsupervised learning, Domain adaptation, Clustering, Dimensionality reduction.

Agradecimientos

En primer lugar quiero agradecer a Daniel Jimenez Carretero y a Ángela Fernández Pascual haber sido unos tutores excepcionales. Gracias por los conocimientos, la comprensión y el tiempo.

Gracias a José Ramón Dorronsoro Ibero por haber sido mi ponente y por la confianza depositada en mi trabajo. También quiero agradecer a María Montoya Sánchez y a toda la unidad de Celómica del CNIC por haberme acompañado durante el desarrollo de todo este trabajo, teniendo paciencia para escucharme y corregirme.

Gracias a la Cátedra UAM-IIC de Ciencia de Datos y Aprendizaje Automático por la ayuda para máster concedida. Gracias al equipo docente del Máster Universitario en Biología Computacional y Bioinformática de la Escuela Politécnica Superior por las clases y los conocimientos impartidos, sobre todo durante este año tan complicado.

Un enorme gracias a mi familia. Gracias mamá, papá y Sergio. Gracias abuelos, tíos y primos por vuestro apoyo y cariño.

A mis ex-compañeros de carrera y a mis compañeros de máster. Ha sido divertido. A mis amigos. A todos, gracias por porporcionarme ayuda y momentos de escape cuando ha sido necesario. En especial, gracias a Carlos por haberme acompañado y apoyado siempre.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and hypothesis

In the era of Artificial Intelligence and Machine Learning, the automated analysis of biological experiments is growing exponentially, making it possible to study larger amounts of high-dimensional data simultaneously. There are thousands of automated pipelines for the analysis of data obtained using different technologies, such as micro-arrays, mass spectrometers, sequencing experiments or, the objective of this work, flow cytometers. In the last 40 years, flow cytometry has become a leader technique in biomedical research, rich in applications, that allows high-resolution single-cell characterization of thousands of cells in a very high throughput way, as they pass through one or more lasers while suspended in a buffered solution and stained with different fluorochromes, allowing the measurement of protein expression levels and intracellular signalling. Letting the identification and quantification of cells types and their function, it is an invaluable tool in many fields, such as virology, immunology or cancer biology, among many others.

The field of flow cytometry is continuously evolving due to the development of new and more complex data acquisition equipment with greater cell phenotyping capacity. The inclusion of more lasers, detectors and fluorochromes has greatly increased the number of parameters that can be simultaneously measured with these instruments. However, the high dimensionality of the collected data represents a challenge for its analysis, traditionally performed manually. Therefore, the field is evolving towards the use of new automatic and unbiased analysis techniques, making use of computational tools and state-of-the-art algorithms for the analysis of big data. As a consequence, the automated and unbiased analysis of flow cytometry data is extremely demanded in the discipline of biomedicine with an increasing interest in methods such as t-SNE for the visualization of high dimensional cytometry data, as well as clustering techniques for the detection of cell populations [1].

However, the automated analysis of data in molecular biology is specially challenging due to variations in the experimental conditions. Several factors can have an influence: the sample management and preparation procedures and equipment variability can have dramatic effects in the quality of data that may hamper its integrated analysis. These non-biological sources of variability, which can be generally named batch effects, cause changes in the collected data that may lead to inaccurate predictions or conclusions. As noted in [2], flow cytometry data is

particularly sensible to batch effects. The experimental conditions in which the data is prepared and acquired greatly influence the subsequent analysis. For example, when applying techniques of dimensionality reduction, such as t-SNE [3] or PCA [4], the scientific community note that data in the embedded space are mainly distributed according to the experimental conditions (batch), instead of being grouped by phenotype. The same sources of error apply to clustering results. This seriously limits the applicability of automatic analysis methods, which, as is commonly the case, joint analysis of data from different experiments is required.

Thus, we hypothesized that domain-adversarial neural networks [5], in the context of autoencoders, could represent an interesting technology to palliate batch effects in flow cytometry data analysis. A standard autoencoder (SAE) [6] is a feed-forward neural network composed of an encoder and a decoder, which are joined by a narrow layer; and they are frequently used to compress data or to obtain a good representation in a space of lower dimension: the latent space or the bottleneck. After least squares training, the encoding weights form a subspace of the principal components input vectors [7]. And not only that, due to their unsupervised nature, they do not require labeled data to be trained, which many times is very expensive or impossible to achieve. However, their main drawback is that the user cannot influence the shape or the distribution of the projected data. Adversarial autoencoders introduce the possibility of including constraints in the encoded representation of data by the use of an adversarial network.

In the context of domain adaptation [8], we propose to train an adversarial autoencoder sensible to the experiment conditions, which we have called domain-adversarial autoencoder (DAE). In the particular case of flow cytometry data, we consider that different experimental conditions (typically, different sample preparation and/or acquisition conditions) define different domains. In that way, the batch effects, whatever their source is, would be contained in the domain. In classification, domain adaptation in neural networks via adversarial learning has shown promising results [5], improving significantly the classification accuracy. Applying domain-adversarial autoencoders, we obtain a double benefit. On the one hand, we acquire *clean* data, free of batch effects, as output of the decoder. On the other hand, we also obtain a representation of the data in smaller dimension as output of the encoder, which can be compared to the t-SNE projection shown in [2]. Moreover, this method allows us to add to the visualization new samples that were not part of the training set, the out-of-sample examples (OOS), without repeating the whole training process from scratch, as it happens with t-SNE and with most of dimensionality reduction techniques typically used in the cytometry field.

The main motivation of this work is to design a method of flow cytometry data analysis capable of palliate batch effects. In particular, our first aim is to review the current state of the art in domain adaptation and present a new unsupervised method, the domain-adversarial autoencoder (DAE). We then focused on DAE and explored its ability to remove batch effects as compared to biological effects contained in flow cytometry data, and developed different metrics to evaluate its capacity. To carry out these objectives, we use a dataset consisting of the flow cytometric analysis of labelled beads with different combination of fluorochromes as an artificial model of cell populations. In addition, we use data obtained from mouse cells, in a more realistic experimental procedure. Even though similar techniques have been applied to analyze data of image cytometry [9, 10], mass cytometry and RNA-seq [11, 12]; we propose here a pipeline to analyze flow cytometry data.

1.2 Objectives and followed approach

The main objective of this work is double. Firstly, as core of this work, we are seeking for the development of a methodology for the correction/removal of batch effects in flow cytometry data, creating a pipeline that allows to analyze samples from different batches (experimental

sources, conditions, etc.) in an integrated manner, thus enabling the direct comparison of data based solely on biological variability. The second objective is to introduce a research line in the field of Artificial Intelligence, focused around domain adaptation in neural networks in the context of unsupervised learning.

With this general purposes, the work is focused on the following topics:

- To review the state-of-the-art methods in adversarial learning, such as generative adversarial networks and adversarial autoencoders.
- To study and understand the existing approaches in domain adaptation via adversarial learning in neural networks.
- To review the automatic pipeline of analysis of flow cytometry data and to understand the problem of batch effects in flow cytometry data.
- To develop a new method, the domain-adversarial autoencoder, which, in the context of unsupervised learning, is able to extract relevant information from flow cytometry datasets, devoid of experimental variability.
- To implement these methods and to validate them in synthetic (beads) and real flow cytometry datasets (mice cells).

1.3 Structure

In this context, this work is structured around four main chapters (two of them dedicated to reviewing the state of the art and two contribution chapters):

1. **Multi-domain problems and deep neural networks**, where we introduce relevant topics of Machine Learning necessary to the understanding of this work. We start reviewing dimensionality reduction methods that have been widely used in biological analysis, such as PCA and t-SNE. Then, we present the concept of unsupervised learning, reviewing clustering techniques and giving a thorough explanation of autoencoders. Finally, we get immerse in the world of adversarial learning in neural networks, visiting the concept of multi-domain problems and existing domain adaptation methods.
2. **Automated analysis of flow cytometry data**. In this chapter we review existing approaches for analyzing flow cytometry data as well as the challenges and the limitations of this kind of analysis. We focus on the challenges originating from batch effect contamination of data and how it limits the interpretation of data from the biological perspective of real biomedical research applications.
3. **Domain adaptation in unsupervised learning**, where we present and implement the domain-adversarial autoencoder (DAE), introducing its advantages and applications in the context of flow cytometry data. We also give some metrics to evaluate the removal of batch effects, which are based in statistical distances that compare the distribution of data in the encoded space.
4. **Experiments**, where we apply the developed methods on flow cytometry datasets. Firstly, with two synthetic datasets that have been generated to test the automatic pipeline analysis of cytometry data. Secondly, with real dataset, containing data from multiple cell types from two different groups of samples (genotypes) to be compared for scientific purposes. For each dataset, we will compute the embedding via t-SNE, a standard autoencoder and

a domain-adversarial autoencoder, comparing the embedding quality and the correction of batch effects.

Finally, we include a chapter of conclusions and further work, where we explore future lines of research.

2

Multi-domain problems and deep neural networks

In this chapter we review the state of the art in dimensionality reduction, unsupervised learning, domain adaptation and adversarial neural networks; establishing the foundations to develop the analysis of the model and problems assessed in this work. In this document, the notation for datasets is $\mathcal{D} = \{x_n\}_{n=1}^{N}$, with $x_n \in \mathbb{R}^m$, where N is the number of samples and m the dimension of data. We say that the dataset is m-dimensional. This chapter is organized as follows:

1. In Section 2.1 we address the problem of dimensionality reduction. Firstly, we review the linear algorithm of Principal Component Analysis (PCA). Due to its importance in the analysis of flow cytometry data, we also review the dimensionality reduction technique known as t-Distributed Stochastic Neighbor Embedding (t-SNE).

2. In Section 2.2 we review the framework of unsupervised learning. Within this discipline, we pay special attention to clustering techniques and autoencoders, a particular case of neural networks that are widely used for data denoising and dimensionality reduction.

3. Section 2.3 introduces the technique of adversarial learning with neural networks. Adversarial neural networks are widely used for data generation, but here we focus in learning with autoencoders, studying the concept of domain adaptation in Machine Learning, and the approach followed to achieve it via adversarial learning.

2.1 Dimensionality reduction

Dimensionality reduction is a field of Machine Learning that aims to transform data from a high-dimensional space into a low-dimensional space retaining meaningful information and maintaining initial relationships. Thus, dimensionality reduction methods are generally used to reduce the initial dimension of data, so the computational load of training a model is reduced; or to extract a limited set of meaningful variables, reducing the influence of noise that may lead to wrong conclusions. Dimensionality reduction is also typically applied before clustering analysis. There are two main techniques to achieve this:

- **Feature selection,** trying to select the most significant variables according to some metric; such as the amount of information they contain.

- **Feature learning,** seeking to create new independent variables by combining, in some manner, the initial features. These methods can be divided into linear and non-linear.

In this work we will focus on algorithms of feature learning, describing in detail the most important methods in the field of flow cytometry data analysis: PCA (linear) and t-SNE (non-linear).

2.1.1 Principal Components Analysis (PCA)

Principal Component Analysis (PCA) [4] is a linear method for feature learning. Basically, its objective is to find a d-dimensional subspace in which the variance of the projected data is maximum, where d is the number of principal components we want to study.

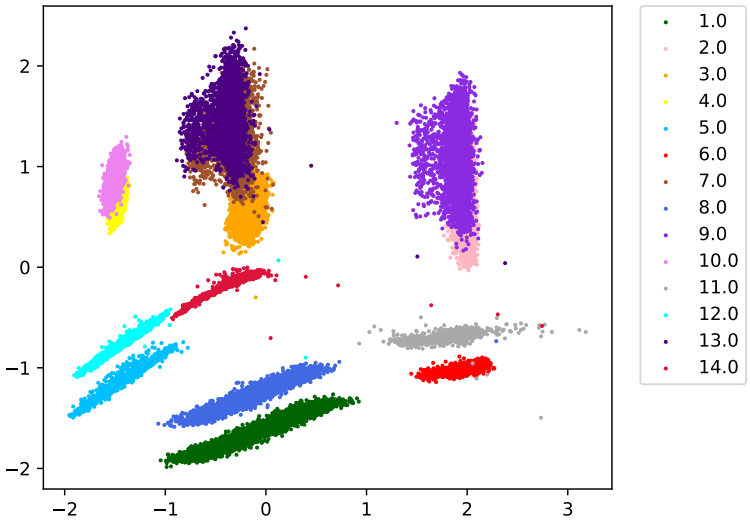

Figure 2.1: 2-dimensional PCA projection of FCM data coloured by class.

The method is based on the spectral analysis of the covariance matrix of the dataset $\mathcal{D}$. Let S be the covariance matrix. As S is symmetric, we can write $S = PDP^T$, where P is the matrix with the eigenvectors of S as columns, which form an orthonormal basis; and D is a diagonal matrix containing the eigenvalues. The eigenvectors that correspond to the largest d eigenvalues, which are the principal components, are used to project the data. This projection can be written as

$$\alpha = P_d^T x \ ,$$

where P_d denotes the matrix containing the first d eigenvectors in columns. Thus,

$$\mathrm{Var}(\alpha) = \mathrm{Var}(P_d^T x) = P_d^T S P_d = D_d \ ,$$

where D_d is a diagonal matrix containing the d largest eigenvalues. So PCA is a method that finds the projection of the data in d dimensions in which the variance is maximum [4]. In addition, as the covariance matrix of the projected data is diagonal, we know that the new features are all independent to one another and that their variance is the corresponding eigenvalue. That is to say, if we set the target dimension d equal to the original dimension m, we will find a projection in which the variables are uncorrelated. However, the main drawback of PCA is its linearity. As we can see in Figure 2.1, the PCA projection of data composed of 14 different populations, which will be treated later in this work (Section 5.1.2), is very deficient. Different classes are intertwined, while, ideally, there should be 14 totally separate groups.

2.1.2 t-distributed Stochastic Neighbor Embedding (t-SNE)

t-distributed Stochastic Neighbor Embedding (t-SNE) [3] is a non-linear method for dimensionality reduction, mainly used for data exploration and visualization. The algorithm tries to minimize the divergence between two distributions: the pairwise similarities of the points in $\mathcal{D}$ and the pairwise similarities in the corresponding points of the embedding. The algorithm consists of two steps:

1. First, t-SNE computes a probability distribution over pairs in $\mathcal{D}$ in such a way that similar points are assigned a higher probability while dissimilar points are assigned a very low probability. For $x_i, x_j \in \mathcal{D}$, their similarity is computed as the conditional probability of selecting x_j as a neighbour of x_i if the probability of selecting a neighbour is proportional to the probability density of a Gaussian centered at x_i.

2. Second, t-SNE computes a similar probability distribution over the low-dimensional map. Then, t-SNE tries to minimize the difference between these conditional probabilities (or similarities) in the original and the projected space. This is done by minimizing the sum of Kullback–Leibler divergence [13] using a gradient descent method. In that way, t-SNE tries to find a projection that preserves the computed similarity in the original high-dimensional data, which is considered a probability distribution, in the low-dimensional mapping.

Recall that the Kullback–Leibler divergence is a measure of how different two probability distributions are. In Figure 2.2 we can see the t-SNE projection of FCM data composed of 14 different populations. Unlike in the projection calculated by PCA (Figure 2.1), points belonging to different classes are now separated in the embedding.

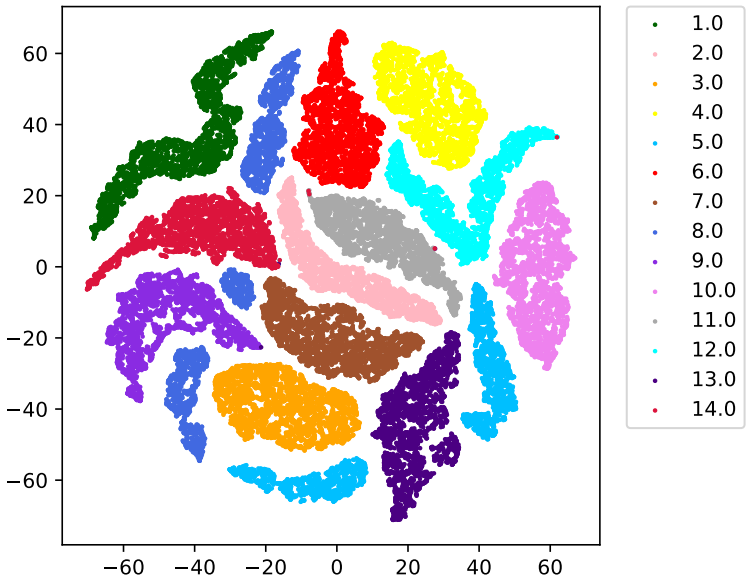

Figure 2.2: 2-dimensional t-SNE projection of FCM data coloured by class.

So the mapping defined by t-SNE attempts to find patterns in data that are preserved in the embedding. This is done identifying similar points (clusters) in the high-dimensional data. The main drawbacks of t-SNE are its high computational demand and the inability to project examples that were not in the training sample (Out-of-sample examples or OOS). That is, it is a sample-dependent method.

2.2 Unsupervised learning

The field of Machine Learning encompasses many disciplines and types of learning. One possibility is to consider the learning strategy depending on the data available to train the algorithm. In this framework, we can make a classification into three types:

- **Supervised learning:** each point $x_i \in \mathcal{D}$ is associated to an output variable y_i and this information is used during the training of the algorithm, trying to find a mapping from the input data to the output variable. Supervised problems can be grouped into classification and regression problems.
- **Unsupervised learning:** labels are not available. The goal for unsupervised learning is to model the underlying structure of the data in order to learn from it. Manifold learning and clustering techniques are a particular case of this learning strategy.
- **Semi-supervised learning:** some data is labeled, so a mixture of supervised and unsupervised techniques can be applied.

In this work we will focus on unsupervised learning techniques, as flow cytometry data will be seldom labeled. In this section we will review cluster analysis and, then, we will give an introduction to autoencoders, one of the principal approaches in unsupervised learning using neural networks.

2.2.1 Cluster analysis

Cluster analysis, also known as clustering, is the problem of identifying objects that, in some sense, can be considered similar. As noted in [14], the clustering problem has been addressed in many contexts, but it is a computationally demanding problem. Still an open problem is to select the number of K groups in which we want to separate data. It is also challenging to asses the performance of a clustering algorithm, as data is unlabeled. Giving an extensive review of clustering algorithms is not the objective of this work. However, in this subsection we will review the most archetypal clustering algorithm, K-means, noting its main disadvantages to analyze flow cytometry data.

A classic algorithm: K-means

Given the number of clusters, K, K-means aims to partition observations in $\mathcal{D}$ in K clusters. First, K centroids are randomly initialized, assigning each point to its closest centroid. K-means minimizes the error function

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\| x_n - \mu_k \right\|^2 , \tag{2.1}$$

where μ_k indicates the k-th centroid and r_{nk} is 1 if the closest centroid to the point x_n is μ_k. The minimization comprises two steps until convergence:

1. First we update r_{nk} as

$$r_{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j \left\| x_n - \mu_j \right\|^2 , \\ 0 & \text{otherwise .} \end{cases}$$

2. The centroids μ_k are updated by minimizing equation (2.1). It is easy to check that the optimal centroid μ_k for the cluster k in the average of the points belonging to that cluster.

In spite of its simplicity, K-means has a strong mathematical background and its results are generally good [15]. However, its main inconvenience is that finding the optimal solution for an arbitrary number of clusters is NP-hard [16]. Generally, in flow cytometry data, we do not know the number of different cell populations in a sample, so we need clustering algorithms that estimate the number of groups. See Chapter 3 for more details.

2.2.2 Standard autoencoder (SAE)

Autoencoders [6] are a particular type of Artificial Neural Networks. They are very popular because of their unsupervised nature and that is the reason why they are widely used for data compression and visualization. They could be considered as a *self-supervised* algorithm because the target is the same as the input data. As shown in Figure 2.3, an autoencoder is composed of two stacked networks: the encoder and the decoder. They are commonly joint by a more narrow layer typically called embedding layer or bottleneck, with dimension d.

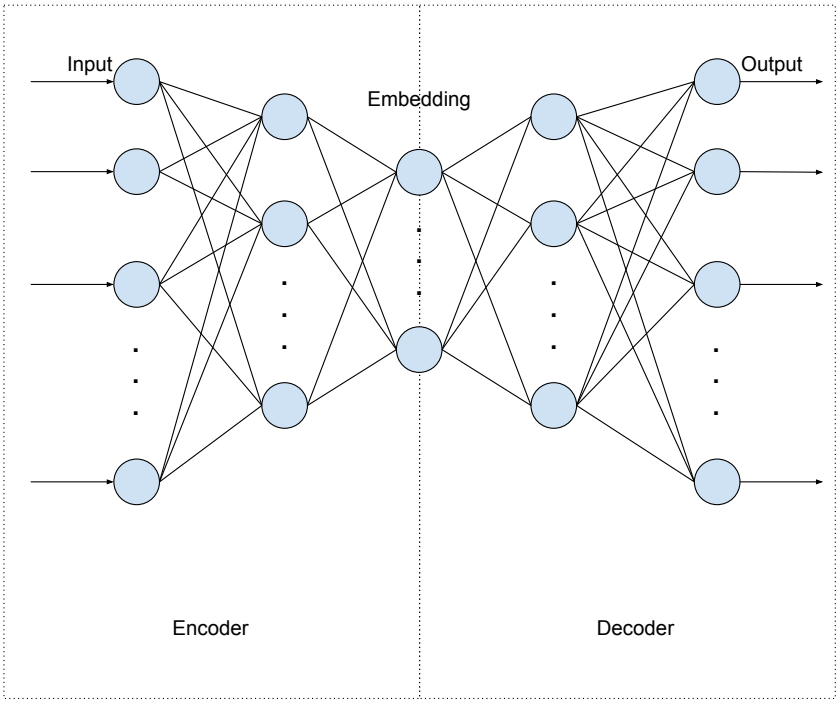

Figure 2.3: Architecture of an autoencoder. Left: encoder. Right: decoder.

The autoencoder is, thus, a function

$$\begin{aligned} o\colon \mathbb{R}^m &\to \mathbb{R}^m \\ x &\mapsto o(x, \Theta)\ , \end{aligned}$$

where m is the input dimension and Θ is the set of weights of the autoencoder. With the objective of reconstructing the input in the output. In order to train an autoencoder, there are three main parts:

- A parametric encoding function E: $\phi = E(x, \Theta_e)$, applied to the input points $x \in \mathbb{R}^m$, where Θ_e is the set of weights of the autoencoder that belong to the encoder part.

- A parametric decoding function D: $\hat{x} = D(\phi, \Theta_d)$, applied to the points in the latent space $\phi \in \mathbb{R}^d$, where Θ_d is the set of weights of the autoencoder that belong to the decoder part.
- A distance function that measures the amount of information lost between the compressed representation in the bottleneck and the decompressed representation. This reconstruction error is used as loss function.

The autoencoder is trained to minimize the reconstruction loss using Stochastic Gradient Descent, computing the gradient with back propagation. Typically, autoencoders are trained to minimize the mean squared error between the input and the output via the following loss function

$$J(\Theta) = \frac{1}{2N} \sum_{i=1}^{N} \|o(x_i, \Theta) - x_i\|^2 + \frac{\nu}{2} \|\Theta\|^2 \ ,$$

where ν controls the regularization term to avoid overfitting. With appropriate dimensionality and sparsity constraints (such as regularization), autoencoders can learn projections that are more interesting than PCA or other basic and linear techniques. Moreover, we obtain an encoding and a decoding function. The encoding function is especially interesting, as it allows us to project into the latent space out-of-sample examples (OOS) that were not in the training set without extra computational cost, in contrast to t-SNE.

Autoencoders have also been retaken in deep frameworks. At first, a standard architecture of an autoencoder was used, for example, in sparse and denoising autoencoders [17, 18]. More recently, more sophisticated versions of autoencoders have been developed. Their goal may not be to learn perfect sample reconstructions but to modify certain properties in the embedding space. Some examples are variational autoencoders [19] and adversarial autoencoders [20]. Due to their importance in this work, we will introduce adversarial autoencoders with more detail in the following section.

2.3 Adversarial neural networks

In adversarial neural networks two networks, a generative and a discriminative network, contest with each other. This framework was firstly introduced by Ian Goodfellow and his group with **generative adversarial neural networks (GANs)** [21]. This kind of models learn to generate new data imitating the distribution of the real samples x in the training set using the two networks previously mentioned:

- The generative network or generator G learns to map from samples z of a latent space with prior $p(z)$ (typically from a Gaussian or a mixture of Gaussians) to a data distribution of interest, which is in the training set (p_{data}).
- The discriminative network or discriminator D differentiates between *fake* data coming from the generator and data from the true data distribution (i.e. the one of the training set).

The generator's objective during training is to *fool* the discriminator, creating realistic data in appearance so the discriminator consider they are not fake or artificial. That is to say, the generator is trained to increase the error rate of the discriminator. The solution to this game is expressed as follows in [21]:

$$\min_G \max_D \left[\mathbb{E}_{x \sim p_{data}} [\log D(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))] \right] \ ,$$

where $\mathbb{E}$ denotes the expected value. The generator G and the discriminator D can be found using alternating stochastic gradient descent in two steps: training D to distinguish between *true* and *fake* samples and then training G to fool the generator with generated samples. Although GANs were initially proposed as generative models in a context of unsupervised learning, their applications have been expanded to semi-supervised [22] and supervised learning [23] and have been applied in a huge number of fields, such as fashion, advertising, drug design or video games.

In this section we will introduce two models that use the proposed adversarial objective and are important for the understanding of the domain-adversarial autoencoders (DAE) proposed in Chapter 4: the adversarial autoencoders and the domain-adversarial neural networks.

2.3.1 Adversarial autoencoders

Adversarial autoencoders (AAE) [20] are probabilistic autoencoders that use the generative adversarial networks to perform variational inference. Although they employ adversarial learning in a different context, we consider them important for the state of the art in the usage of adversarial networks in the context of autoencoders.

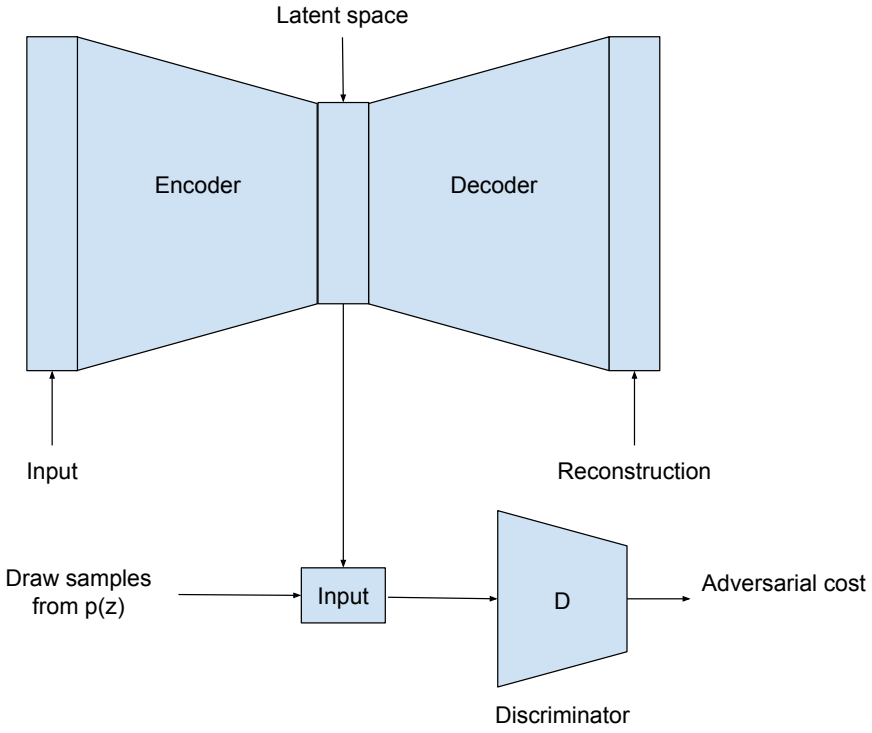

Figure 2.4: Architecture of an adversarial autoencoder. Top row is a standard autoencoder that minimizes the reconstruction error of an input given its latent representation. The second row is a discriminator which aim is to discriminate if samples come from the embedding of the autoencoder or from the prior distribution $p(z)$.

Adversarial autoencoders are able to impose a certain distribution in their latent space using a discriminative network (Figure 2.4). Let x be the input and z the latent vector of the autoencoder, with probability distribution $q(z)$. Assume that $p(z)$ is the distribution desired in the coding space. The adversarial network guides $q(z)$ to match $p(z)$ while the autoencoder endeavors to minimize the reconstruction error. As in GANs, both the discriminative network and the autoencoder are trained simultaneously with iterative stochastic gradient descent in two steps:

1. Reconstruction phase. The autoencoder updates its weights minimizing the reconstruction error of the input data.

2. Regularization phase. First, the discriminator is updated to distinguish real and generated samples from $p(z)$. Then, the weights of the encoder are updated to confound the discriminative network.

After the training procedure, the decoder is a generative model that maps the prior $p(z)$ to the distribution of the data, p_{data}. Some of the applications of adversarial autoencoders are unsupervised clustering, dimensionality reduction and data visualization [20]. Even though adversarial autoencoders are not focused in domain adaptation or removal of batch effects, which are the main objectives of this work, we consider them important because of the way they employ an adversarial network to modify some properties of the latent space of the autoencoder, in this case, to fit a desired distribution. Therefore, it is necessary to define $p(z)$, which means that we need previous information that reduces the unsupervised scenery. In our application, we do not have prior information of this type. Thus, a similar domain-based approach will be followed to remove the batch effects in the experiments of this project.

2.3.2 Domain-adversarial neural networks

Domain adaptation [8] is a Machine Learning problem where training and testing data proceed from similar but different distributions. The key idea is to train an algorithm that cannot distinguish between domains. In this manner, we typically seek the generalization between two similar problems in which one of them is not labeled. For example, we can have labeled data of sentiment analysis from book reviews, but we may want to generalize it to unlabeled data from films reviews [5].

There are several approximations to address this problem. Specifically, a variant of denoising autoencoders [17], the marginalized stack autoencoder [24], have demonstrated state-of-the-art performance. By learning a robust representation against noise, they learn stable features across domains, allowing domain transfer and classifier learning jointly. In the context of adversarial learning, domain-adversarial neural networks (DANN) [5] propose to control the stability of representation explicitly in the neural network with the incorporation of an adversarial network. Thus, DANN simultaneously learn the desired function of classification and try to predict domain membership. With extensive experiments, they show that this approach outperforms standard neural networks and SVMs (Support Vector Machines) in classification problems.

Hence, in [5], authors propose DANN as a method to address classification problems in which there are a *source domain* $\mathcal{D}_S$ and a *target domain* $\mathcal{D}_T$. In this context, the classification algorithm is provided with a labeled source sample $S = \{(x_i, y_i)\}_{i=1}^{n} \sim \mathcal{D}_S$ and a target sample $T = \{x_j\}_{j=1}^{n'}$, which is unlabeled. Being $N = n + n'$ the total number of samples, the objective is to learn a good classifier for T, although we have no information about $\mathcal{D}_T$. In order to confront this problem, there are several approaches. Domain-adversarial neural networks are focused on the minimization of the $\mathcal{H}$-divergence [8], as they show that the learning algorithm should minimize a trade-off between the correct classification of the source and the $\mathcal{H}$-divergence. As noted in [8], controlling the $\mathcal{H}$-divergence can be done finding a representation where both S and T are as equal as possible.

In that way, they claim that if a neural network is able to generalize between domains,the internal representation among hidden layers cannot contain information about domains. Domain adaptation is achieved by adding a domain classifier (Figure 2.5) connected to the layer containing the hidden representation via a gradient reversal (*flip*) layer. Here comes the adversarial game: the feature extractor must find a hidden representation that is good enough to

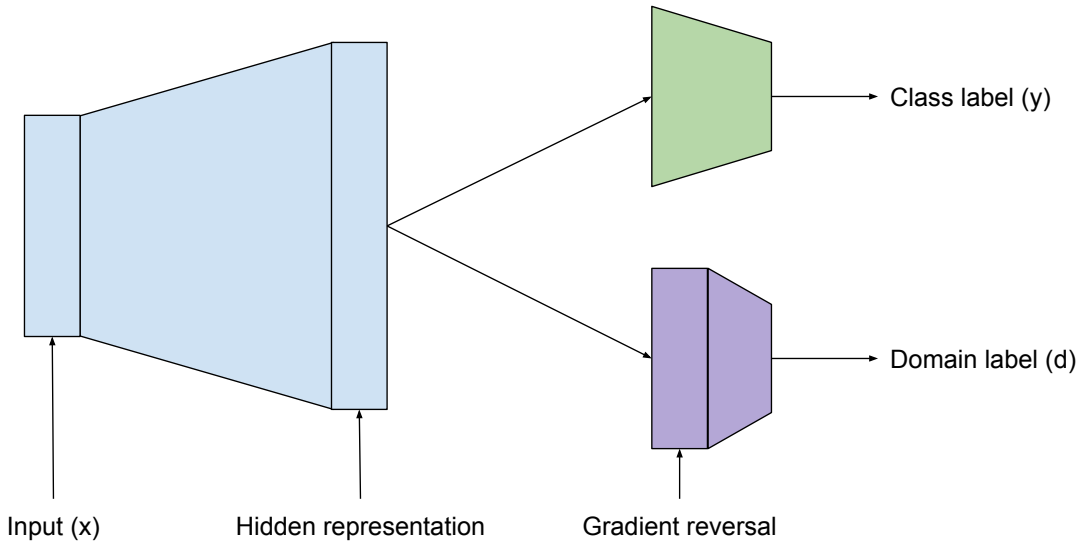

Figure 2.5: Architecture of a domain-adversarial neural network. It comprises three neural networks: a feature extractor (blue), a label predictor (green) and a domain predictor (purple).

predict the class labels of S at the same time it tries to *fool* the domain predictor, so it is not able to distinguish between examples coming from S and T. Unlike in GANs and adversarial autoencoders, optimization is carried out via stochastic gradient descent in one step thanks to the appearance of the *gradient reversal layer* (GRL). During backpropagation, the flip layer multiplies the gradient by a negative constant, encouraging features in the hidden of source and target domains to be similar.

Adversarial learning methods are a promising approach to training robust deep networks, generating complex samples across diverse domains. However, this approach can only be applied when labeled data comes from the source domain, while target data is distributed with a similar but different distribution. Therefore, this methodology requires labeled data, which is very hard to obtain. Despite this, we do not have any evidence of domain-adversarial adaptation with an adversarial network in unsupervised learning techniques, such as the autoencoders. Not only that, with domain-adversarial neural networks we have the explicit constraint of having a source and a target domain, while we also might be interested in training all at once with data coming from several domains, many times more than two domains. All these problems will be addressed in Chapter 4.

3

Automated analysis of flow cytometry data

This chapter will be devoted to the revision of automated analysis of flow cytometry data. Being one of the most used techniques of single-cell analysis, flow cytometry (FCM) allows the phenotypic study of cell populations. Due to a technological revolution, automating flow cytometry analysis is especially important because of the large number of parameters and data that can be measured simultaneously. In the context of unsupervised learning, we are seeking to review the main clustering and visualization techniques applied in flow cytometry experiments. The chapter will be organised as:

- In Section 3.1 we will study the main properties of the data obtained by flow cytometers as well as the preprocessing steps needed to its proper analysis. Thus, we will introduce the biexponential transformation, which is the normalization technique more frequent in flow cytometry data.
- In Section 3.2 we will review the most common approaches followed to the cluster analysis of flow cytometry data. We will introduce the algorithm *Phenograph* and the $F1$-score, metric that will be used to evaluate the performance of the cluster analysis in testing data that is manually labeled.
- Section 3.3 will contain standard procedures used for visualization in two or three dimensions of flow cytometry data, which are typically multi-dimensional.
- Finally, in Section 3.4, we will introduce the problem of batch effect in flow cytometry data, difficulty that we aim to solve in this work. We will show some examples in which batch effect does not allow the proper analysis of the data, as differences among experiments do not permit to see differences in cell populations.

3.1 Flow cytometry (FCM)

Flow cytometry (FCM) is a technique that allows high-resolution single-cell characterization of thousands of cells in a very high throughput way, as they pass through one or more lasers while suspended in a buffered solution (Figure 3.1). The cells are stained with different fluorochromes, allowing the measurement of protein expression levels and intracellular signalling, together with

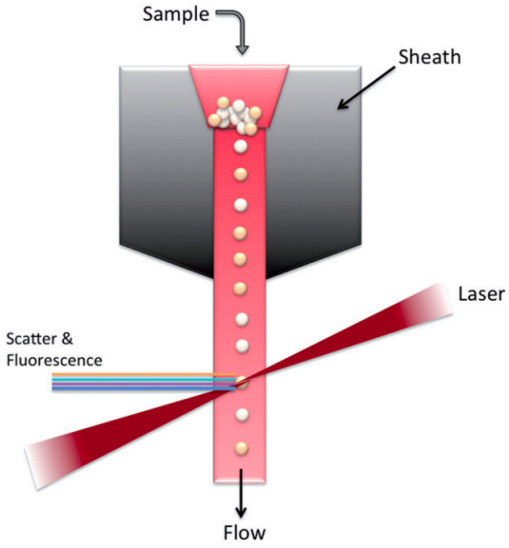

Figure 3.1: The underlying working principle of a flow cytometer. Image from [25].

size and shape information. Because of that, the spectrum of applications of FCM is huge, starting from clinical diagnosis to biomedicine or molecular biology. The main characteristics of FCM are:

- The analysis is multi-parametric and simultaneous. The analyzable parameters are:
  - Those related with physical characteristics of the particle, such as size and complexity. Therefore, depending on how light is modified when passing through cells, they can be discriminated.
  - The fluorescence associated to the particle.
- The analysis is performed on individual particles inside a complex population, such as fluorochrome coupled antibodies or fluorescent proteins and molecules.
- The analysis is carried out at speeds from 500 to 4000 particles per second, allowing us to analyze a very large number of cells which enables the identification of populations represented at very low frequency.
- Particles of sizes 0.5-100 μm can be analyzed.

As shown in Figure 3.2, a cytometer is composed of different subsystems. First of all, the fluid system where the sample is in suspension and flowing through the circuit thanks to the surrounding liquid. Then, the laser beam will strike, so we can detect fluorescence at different wave-lengths, which is detected by an electronic network composed of several detectors connected to a computer. The fluid passing through the lasers is such a fine beam where "only one cell fits", allowing analysis of individual cells. Finally, the electronic network converts the signal in digital data that can be stored and analyzed with a computer. As introduced before, we can measure two types of parameters with a flow cytometer:

- Intrinsic or structural factors, for which no reagents or probes are needed. The most common are:
  - Cell size (side scatter or FSC).

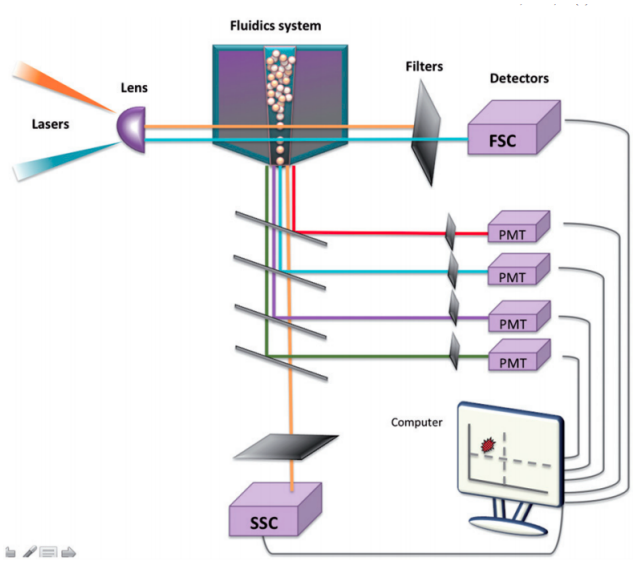

Figure 3.2: Signal processing and detection. Image from [25].

  - Cell complexity (side scatter or SSC).

- Extrinsic factors, that need the labelling of cellular structures with fluorochromes.

Once the electronic network has converted the signal in digital data, it is stored in the computer. Before its analysis, FCM data must be corrected via background fluorescence subtraction and, more important, compensation procedures to correct the spectrum overlap between different fluorochromes.

In order to perform the analysis, FCM users typically find one-parameter histograms or two-parameter histograms, also called dot plots. Histograms corresponding to each parameter can be analyzed using statistical tools to calculate the percentage of cells that fluoresce in a specific way and intensity. Nowadays, these analysis require expert knowledge and *manual work*; this can be done when the number of available parameters is restricted, most commonly between 2 and 10.

However, in the last times, FCM has undergone a technological development, increasing the number of parameters that can be analyzed in a cell [1, 26]. Thus, the number of available detectors has increased, reaching as many as 30 in some cytometers. This has driven the progress in computational tools that enable semi-automatic analysis and interpretation of high-dimensional data from FCM. Thanks to its power and versatility, the applications of flow cytometry are large. In particular, this work is focused in the phenotypic profiling of the sample, with the objectives of identifying cell populations in the sample and finding possible alterations in known cell populations. In this work, we will analyze synthetic and real flow-cytometry data. Each dataset is explained with more detail in Section 5.1.2.

3.1.1 Preprocessing of FCM data

The biexponential (logicle) transform [27] is an effective method that provides several advantages for analysis and visualization of FCM data. In FCM applications, fluorescence signals can range to essentially zero. In addition, some cell populations will have low means and even negative data values, specially after compensation. In these particular situations, standard logarithmic transformation, which is also typically used in FCM data, fails to adequately display

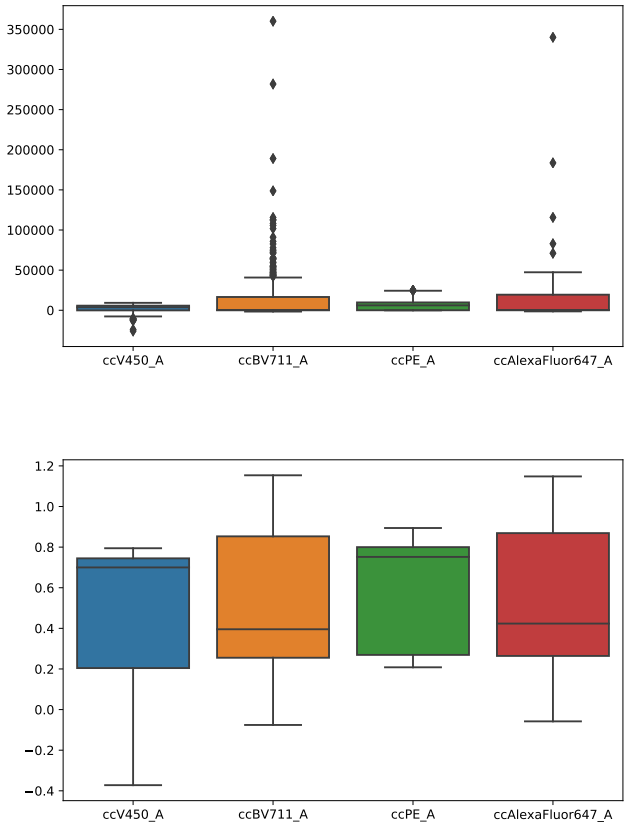

Figure 3.3: Example of FCM data before (up) and after (down) applying the biexponential transform.

cell populations, specially when comparing populations with low levels of fluorescence versus high-levels-of-fluorescence populations. For these reasons, FCM data have been difficult to interpret.

Logicle functions are a generalization of the hyperbolic sine function. By applying a biexponential transform to the data, we obtain a more accurate visual representation of fluorescence units in the low range of the scale as compared to the higher range of the scale. This effect is shown in Figure 3.3. The upper figure shows the boxplot of FCM data before applying the transformation. We can observe very unbalanced data, with extreme outliers. In the figure below it is easy to see that the biexponential transformation works well on this data, softening the influence of outliers. The authors of this transformation [27] had a triple objective:

- The transformation is logarithmic for large data values.
- The function becomes linear near zero and extends to negative values.
- The transition between the logarithmic and the linear regions is smooth.

They conclude that a generalization of the hyperbolic sinus function, which they call logicle function, can best fulfill the criteria above. The expression for the logicle scaling function is:

$$S(x;w) = Te^{-(m-w)}\left(e^{x-w} - p^2 e^{-(m-w)/p} + p^2 - 1\right) \text{ for } x \geq w\ , \tag{3.1}$$

where T is the top scale data value, $w = 2p \ln p/(p+1)$ is the range of negative and linearized data and m is the range of visualization. Equation (3.1) is for the positive data that fulfills $x \geq w$. In the negative region, where $x < m$, authors enforce symmetry using the logicle function for the value $w - x$. In $x = w$, the derivative of Equation (3.1) is zero, so it is the linear region of the transformation. These parameters are estimated from data.

3.2 Clustering with flow cytometry data

As a general rule, FCM data is not labeled. We do not know what type of cells compose a given sample, and, in general, how many different populations there are neither. Because of that, unsupervised learning techniques are popular when analyzing FCM data. Clustering algorithms are of special interest, since they allow us to identify different cell populations (which would be the identified clusters). Moreover, a most common application in flow cytometry deals with finding sample-based differences in the density of the populations that follow a certain biological perturbation and may be absolutely unexpected.

Therefore, in order to analyze FCM data, it is important to find an effective clustering algorithm, capable of handling large volumes of data and of identifying in an unbiased manner the different populations present in the sample, which are in principle unknown, not only their number, but also their characteristics. In [2], authors select as the default clustering method Phenograph [28], as it obtained the best confident interval of $F1$-scores when analyzing conventional FCM data. In this subsection we will describe this method and the $F1$-score as a possible metric to be used.

3.2.1 Phenograph

Phenograph [28] is a computationally efficient clustering method designed for single-cell data. It creates a graph (or network) that represents phenotypic resemblance among cells. Then, Phenograph identifies different cell populations in the graph using modularity optimization. The main advantages of Phenograph are:

- Phenograph works well on large samples, producing high-quality results without downsampling.
- Phenograph automatically estimates the number of cell populations in the sample (i.e. the number of clusters).

Phenograph builds the graph in two steps. Firstly, it builds a graph in which each cell (data point) is a node connected to its k-nearest (euclidean distance) neighbours. Note that we need to set k for this step, but the authors claim that the final result is not very influenced by this choice. In the second step, the graph is refined building a weighted graph in which the weight between nodes i and j is their Jaccard similarity coefficient, which is

$$W_{ij} = \frac{|v(i) \cap v(j)|}{|v(i) \cup v(j)|} ,$$

where $v(i)$ is the set of k-neighbors of the node i defined before. This metric is maximum for nodes with the same set of neighbours and tunes automatically the number of neighbours k, as edges in dense regions are reinforced and edges in sparse regions are penalized. The term modularity in a graph refers to the presence of modules that are densely intraconnected. The task is to find a partition of the graph that captures its modular structure. The problem

of finding the partition that maximizes modularity is NP-complete. However, the Louvain community detection method [29] is a good approximation. In particular, Phenograph runs multiple random restarts of the Louvain method and chooses as solution the partition with the maximum modularity.

3.2.2 $F1$-score

Assessing the quality of the results obtained by a clustering algorithm is still an open problem, as we do not have labels that can be employed to define the accuracy or the error. Sometimes, we can use previously labeled data (many times, synthetic) to determine if the algorithms are working properly. Even though this situation does not reflect real-life problems, this approximation allows us to make decisions, hoping that the behaviour of the algorithm will be similar with new unlabeled datasets.

In this work, we will use some synthetic datasets to evaluate the performance of the algorithms tested. We will focus on the $F1$-score [30], which is a measure of the accuracy of a test. In binary classification, the $F1$-score is the harmonic mean of the precision and recall, being 1 the best value and 0 the worst. Thus, the $F1$ score is defined as

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} ,$$

where:

- *Precision* is the number of true positives divided by the total positives.
- *Recall* is the number of total positives divided by the number of samples that should be positive.

In multi-label problems, the $F1$-score is the average of the $F1$-score of each class. An important issue to mention is the way we calculate the cluster-class assignments. One approach is to select, for each class, the cluster that maximizes the $F1$-score for that class. But this does not prevent the same cluster from being selected for multiple classes. For this reason, we use the Hungarian assignment algorithm [31], which finds a mapping that maximizes the sum of $F1$-scores of all the reference classes, ensuring that for each reference class only one cluster is selected. Thus, there can be clusters that do not "match" with any reference class. This decision makes the evaluation very strict (although also more fair), because in order to obtain a good $F1$-score, both the number of clusters and their precision are very important, so that the existence of overclustering is greatly penalized.

3.3 Dimensionality reduction in flow cytometry data

In order to visualize clustering results and cell populations in FCM data, we can use a dimensionality reduction method that projects data into a 2-dimensional space. As mentioned before, FCM data is typically multidimensional. Not only that, the number of channels tend to increase with the development of technical aspects. Even though Phenograph is applied to the original fully dimensional FCM data, it is interesting to provide the user with a 2-dimensional map to achieve a better visualization and understanding. In addition, 2-dimensional projections are the standard display procedure, since when the number of measured parameters is increased, the number of paired combinations of parameters also increase, making visualization a tedious task, which is biased to the order in which these projections are displayed.

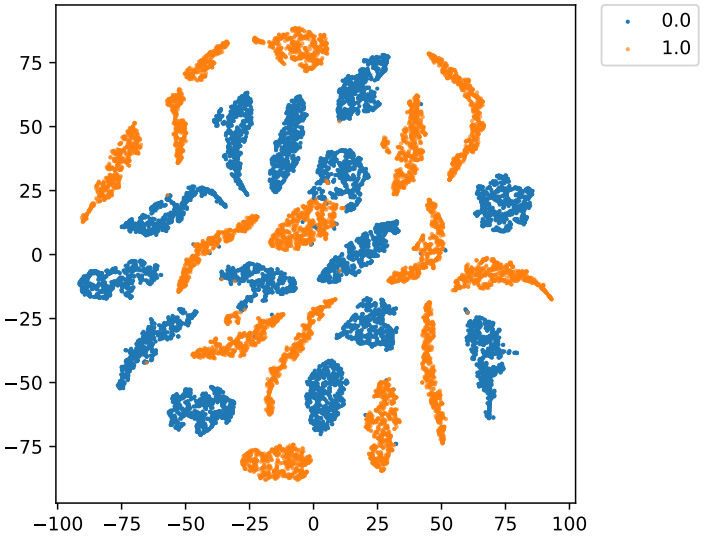

Figure 3.4: The influence of batch effect in FCM data: t-SNE projection of equivalent FCM data obtained from two different cytometers. Colors distinguish the cytometer used in the experiment (i.e. each domain).

With this purpose, authors in [2] select the t-SNE projection. In general, t-SNE is a reliable choice, as it handles non linear data and preserves both global and local structure, so it has been widely used for visualization of FCM data in recent years [32]. However, the main disadvantages of applying t-SNE to FCM data are the following:

- t-SNE depends on data and does not provide a function to project new data. This means that it is difficult to embed into the latent space examples that were not in the training sample provided to t-SNE. They are the out-of-sample examples (OOS examples).
- t-SNE is computationally expensive. This is specially unfavourable when treating with FCM data, as we deal with large volumes of data, so we will need an strategy for down-sampling. This also implies to define an strategy of up-sampling FCM events.
- t-SNE is very sensible to batch effects, which will be introduced in Section 3.4.

In this work we will try to mitigate some of these effects applying Adversarial Autoencoders to FCM data, as we will see in the following chapters.

3.4 The problem of batch effect in flow cytometry data

As we have previously mentioned, one of the biggest challenges in automating the analysis of problems with a biological background is their dependence on experimental conditions. Batch effects are non-biological factors: they are the unpredictable technical differences we find when biological samples are collected, processed and measured. Thus, avoiding the influence of batch effects is specially challenging when integrating data from different experiments, which have been prepared and/or measured in different days or nonidentical conditions. The main objective of this work is to perform batch normalization using Machine Learning. Remember that domain adaptation in Machine Learning tries to infer properties from data that comes from similar but slightly different distributions. One way of achieving this is to train algorithms that are not capable of distinguishing data according to their domain. In order to achieve batch correction,

we will consider that the information about the experimental conditions is contained in the domain.

In Figure 3.4 we can see an example of the influence of batch effects in the analysis of FCM data. The projections appear colored with two different colors: they are data from equivalent beads but acquired on two different machines. The first kit is a conventional cytometer (FORTESSA) and the second kit is a spectral cytometer (SP6800). The samples were prepared and acquired in the same way and on the same dates, so the differences between the two are almost exclusively due to the use of a different cytometer. This is considered as batch effect in our particular example. As we can see, the t-SNE projection is very sensitive to these effects: data from different domains hardly intersect with each other. Our goal is to obtain data from different domains (i.e. different experimental conditions) completely overlapped for the same populations.

Some previous works tried to mitigate batch effects using a specific network architecture or custom loss functions. In [11], authors use autoencoders to perform batch normalization in mass cytometry and RNA-seq data. In this case, batch normalization is achieved in the embedding space using regularization with the Maximum Average Discrepancy (MMD) [33], obtaining good results. However, this method is restricted to two-domain problems. We will see in Chapter 4 that our approach, which is based on the concept of adversarial network, is a relatively easy architecture which, in addition, is general for an arbitrary number of domains.

4

Domain adaptation in unsupervised learning

In previous chapters we described fundamental concepts detailing methods and algorithms that are needed to analyze FCM data and its challenges and limitations, which help us to fully understand the proposal of this work: the domain-adversarial autoencoder (DAE). Thus, this section is focused on the algorithm developed to achieve domain adaptation (i.e. batch correction) with autoencoders incorporating an adversarial network. Regarding definitions, from the point of view of Machine Learning, the DAE is an algorithm for *domain adaptation*. However, concerning flow cytometry data, the DAE is applied with the objective of correction of batch effects (to simplify, batch correction). In this and following chapters, we use these terms indifferently, as we identify domains with data acquired in different experimental conditions. This chapter is divided in two sections that cover the following topics:

- In Section 4.1, we introduce the architecture of a domain-adversarial autoencoder (DAE), explaining the adversarial network to perform domain adaptation or batch correction. Training adversarial networks is always a subtle problem so we also give a detailed schema of the technique used in this work.
- Section 4.2 presents some approaches to assess the performance of batch normalization. Since we are facing an unsupervised learning problem, evaluation techniques are always an open and complex problem. In this section we present an evaluation proposal that can be applied in a generic way, regardless the number of considered domains.

4.1 Domain-adversarial autoencoder (DAE)

The domain-adversarial autoencoder (DAE) is the method proposed in this work to achieve domain adaptation in an unsupervised-learning context. As we have seen before (Section 2.2), an autoencoder is composed of two parts: an encoder and a decoder. The encoder is the net that projects the data into the latent space or embedding. The decoder reconstructs the original data given the latent representation. Domain-adversarial autoencoders are based in the following principle: if the autoencoder is able to generalize between domains, the embedding cannot contain information about them. If the embedding does not contain information about the domain, the reconstruction of the decoder should be domain-independent.

4.1.1 DAE architecture

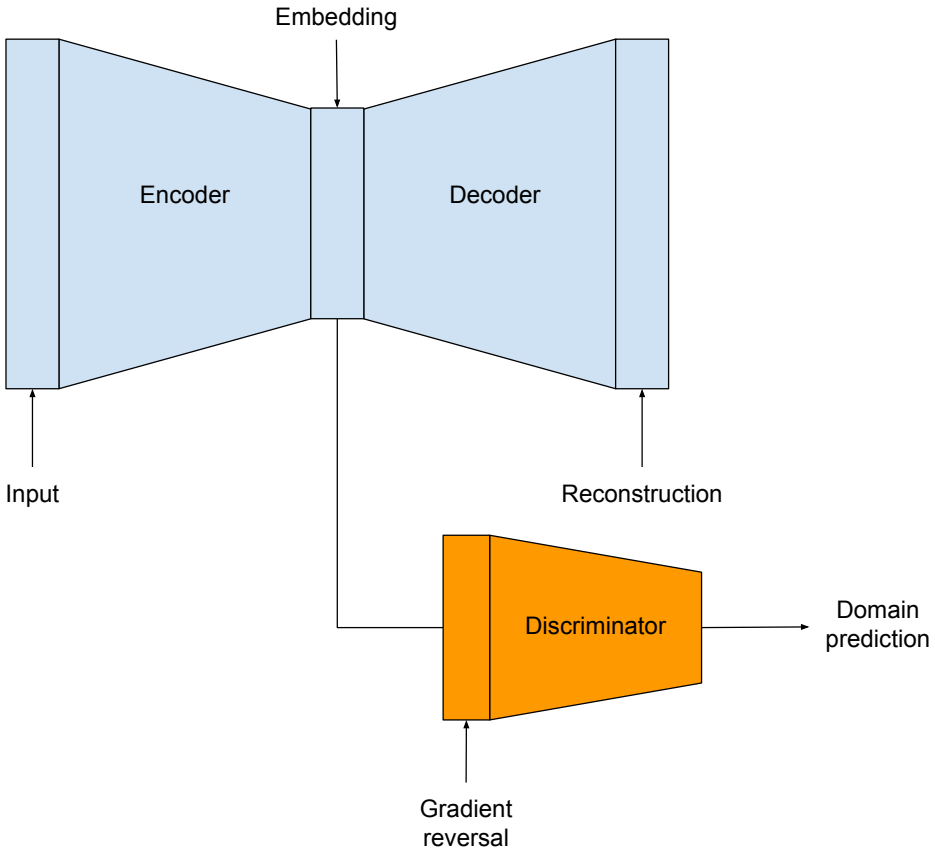

Figure 4.1: Architecture of a domain-adversarial autoencoder. Top row is a standard autoencoder (blue) that minimizes the reconstruction error of an input given its latent representation. The second row is a discriminator (orange) which aim is to discriminate to which domain the samples belong.

A DAE is a model composed of two neural networks, see Figure 4.1. The first one is a standard autoencoder, which, as explained before, is a symmetric network with the objective of reconstructing the input at the output. The second one is the discriminator or domain classifier, which is the net in charge of domain adaptation. The purpose of this is to discriminate the domain from which the data comes given its embedding in the latent space. Thus, a DAE is a composition of three functions:

- A parametric encoding function E: $\phi = E(x, \Theta_e)$, applied to the input points $x \in \mathbb{R}^m$, where Θ_e is the set of weights of the autoencoder that belong to the encoder part.
- A parametric decoding function D: $\hat{x} = D(\phi, \Theta_d)$, applied to the points in the latent space $\phi \in \mathbb{R}^d$, where Θ_d is the set of weights of the autoencoder that belong to the decoder part.
- A parametric discriminating function δ: $\delta(\phi, \Theta_\delta)$, applied to the points in the latent space $\phi \in \mathbb{R}^d$, where Θ_δ is the set of weights of the discriminator.

In this case, the encoder E needs to find a good representation of the data in a d-dimensional space that allows the decoder to perform a good enough reconstruction while trying to *fool* the discriminator and, thus, removing the information about the domain in the latent space. The

adversarial game is solved introducing a gradient reversal layer (flip layer) between the encoder E and the discriminator δ. This layer is the identity function during forward propagation. However, during back propagation it multiplies its input by -1. Thus, during back propagation the output of the gradient reversal layer is leading to the opposite of gradient descent in the encoder with respect to the discriminator.

4.1.2 Training a DAE

In order to be trained, a DAE must be provided with a "labeled" sample of N events, $S = \{(x_i, d_i)\}_{i=1}^{N}$, where x_i is the data point and d_i is the domain from which this point comes. Note that d_i is not a true label in the sense of classification problems, it is just a vector containing a domain identification. For example, assume we have data from three different experiments in a laboratory. Then, $d_i \in \{0, 1, 2\}$. This type of label does not incorporate expert knowledge, as the definition of different domains (experiments in this case) is objective.

Rigorously, a DAE is a network trained in an adversarial context, so we need to define two loss functions. We note the reconstruction loss and the domain loss for the sample (x_i, d_i) by:

$$J_r^i(\Theta_e, \Theta_d) = J_r\left(D(E(x_i; \Theta_e); \Theta_d), x_i\right) \quad \text{and} \tag{4.1}$$

$$J_d^i(\Theta_e, \Theta_\delta) = J_d\left(\delta(E(x_i; \Theta_e); \Theta_\delta), d_i\right) \tag{4.2}$$

respectively. J_r and J_d are specified by the user. Training a DAE consists in optimizing

$$J(\Theta_e, \Theta_d, \Theta_\delta) = \left(\frac{1}{2N}\sum_{i=1}^{N} J_r^i(\Theta_e, \Theta_d) + \frac{\nu}{2}\left\|\Theta_e\right\|\right) - \left(\frac{1}{2N}\sum_{i=1}^{N} J_d^i(\Theta_e, \Theta_\delta)\right) , \tag{4.3}$$

where ν controls the regularization term to avoid extremely large values in the embedding space. Thus, the objective is to find a set of weights that are optimal in the classical adversarial game

$$(\Theta_e^*, \Theta_d^*) = \underset{(\Theta_e, \Theta_d)}{\operatorname{argmin}}\ J(\Theta_e, \Theta_d, \Theta_\delta) , \tag{4.4}$$

$$\Theta_\delta^* = \underset{\Theta_\delta}{\operatorname{argmax}}\ J(\Theta_e, \Theta_d, \Theta_\delta) . \tag{4.5}$$

The encoder E, the decoder D, and the discriminator δ can be found with standard stochastic gradient descent (SGD) in two stages:

- Train the autoencoder to minimize the reconstruction loss.
- Train the discriminator to distinguish the domain of the samples.

Algorithm 1 contains the pseudocode of this learning methodology (in this case, batch refers to the amount of examples used in each iteration of training in the neural network). In general, we believe that two good choices for the reconstruction and domain losses are the mean squared error or MSE, and the categorical cross-entropy, respectively. The mean squared error is a classical choice in regression problems (deep inside, an autoencoder is a regression problem in which the input and the output are the same) and the categorical cross-entropy is one of the most used losses to train multi-class classification networks. Specifically, our loss functions would be:

$$J_r(\Theta_e, \Theta_d) = \left(\frac{1}{2N}\sum_{i=1}^{N} \left\|D(E(x_i\Theta_e), \Theta_d) - x_i\right\|^2 + \frac{\nu}{2}\left\|\Theta_e\right\|\right) \text{ and}$$

$$J_r(\Theta_e, \Theta_\delta) = \left(\frac{1}{2N}\sum_{i=1}^{N}\sum_{c=1}^{l} y_{x_i,c}(\Theta_\delta)\log p_{x_i,c}(\Theta_\delta)\right) ,$$

where l is the number of considered domains, y is a binary indicator for the domain to be correctly predicted for the observation x_i and p is the predicted probability for observation x_i to belong to the domain c. If the problem is composed of two different domains, the standard binary cross-entropy is also a good choice. Regarding activation functions, we propose to use the rectified linear units (ReLU), in every hidden layer; the linear function in the last layer of the autoencoder and the softmax function in the last layer of the discriminator (which are also quite standard set-ups both in regression and classification problems). Concerning the architecture of the networks, the learning rate and the optimizer, we recommend the user to perform a grid search to find the optimal combination. As the grid search cannot be done training the DAE (note that we are not just minimizing the reconstruction error, we have a much more complex loss function), we decided to do it with an standard autoencoder and use the same configuration later for the DAE.

Algorithm 1: DAE - training update

input: Samples $S = \{(x_i, d_i)\}_{i=1}^{N}$
Encoder E, weights Θ_e
Decoder D, weights Θ_d
Discriminator δ, weights Θ_δ
Number of training epochs n_e
Batch size m

begin
  for $i \in \{1, 2, \ldots, n_e\}$ **do**
    $B \leftarrow$ Sample minibatches of m examples $\{(x^1, d^1), \ldots, (x^m, d^m)\}$ from S
    for $B_i \in B$ **do**
      $J_r \leftarrow$ Compute reconstruction loss between B_i and $D(E(B_i))$
      Update weights of E and D via SGD of J_r
      $J_d \leftarrow$ Compute domain loss in B_i
      Update weights of δ via SGD of J_d
      Update weights of E via SGA of J_d
    end
  end
end

Clarifications:
SGA: stochastic gradient ascent.
SGD: stochastic gradient descent.
Sample minibatches: sample $\lfloor N/m \rfloor$ batches from S.

4.2 Assessment of batch normalization

The assessment of domain adaptation or batch normalization is still an open problem. There are no standard ways of measuring the error or the success of the algorithms. In this work we have argued that if there is no information about the domains in the latent space learnt by the DAE, then we should not see the effects of batch normalization in the reconstruction given by the decoder when we train with equivalent data whose differences are due to the domain, batch or experiment. Thus, we propose to use a statistical distance that compares distributions among domains in the latent space of the DAE. The reasoning is as follows: if the distributions of the data given the domains are statistically indistinguishable, then there cannot be domain information. Thus, we would be able to evaluate the capacity of the algorithm to carry out the domain adaptation with two important advantages:

- We are using a non-parametric statistical notion of distance, avoiding possible biases in the models.
- We are obtaining a number that objectively serves to evaluate the quality of the batch normalization, instead of simply evaluating it in a visual way, which is also clearly subjective and dependent on factors such as the scale or the quality of the drawing.

4.2.1 Multidimensional KS distance

The Kolmogorov-Smirnov (KS) [34] statistic is a notion of statistical distance between the empirical distribution function of a sample and a reference distribution $F(x)$. The empirical distribution F_n for n independent and identically distributed (i.i.d.) ordered observations $\{X_i\}_{i=1}^n$ is

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,x]}(X_i) \ , \tag{4.6}$$

where $I_{(-\infty,x]}$ is equal to 1 if $X_i \leq x$ and 0 otherwise. The KS distance can be computed as

$$D_n(x) = \sup_x |F_{1,n}(x) - F(x)| \ .$$

The Glivenko-Cantelli theorem [35] ensures that if $F(x)$ is the distribution from where the sample comes from, then $D_n \to 0$ almost surely when $n \to \infty$.

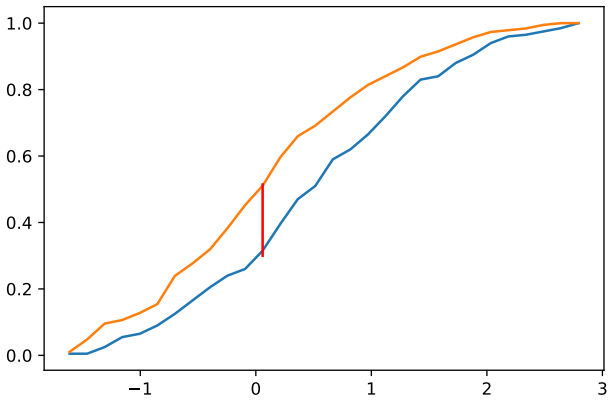

Figure 4.2: Example of the two-sample KS distance. Orange and blue lines are the empirical distribution functions of two samples randomly generated. The red segment represents the corresponding KS distance between them.

The two-sample KS distance quantifies the statistical distance between the empirical distribution of two samples, being one of the most useful and general non-parametric methods for comparing samples. Suppose we have two sets of ordered i.i.d. observations, $\{X_i\}_{i=1}^n$ and $\{Y_j\}_{j=1}^m$. In this case, the two-sample KS distance is:

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)| \ ,$$

where $F_{1,n}(x)$ and $F_{2,m}(x)$ are the empirical distribution functions of the samples, defined as in Equation (4.6). Figure 4.2 contains the illustration of this value, where the two-sample KS distance is the length of the red segment.

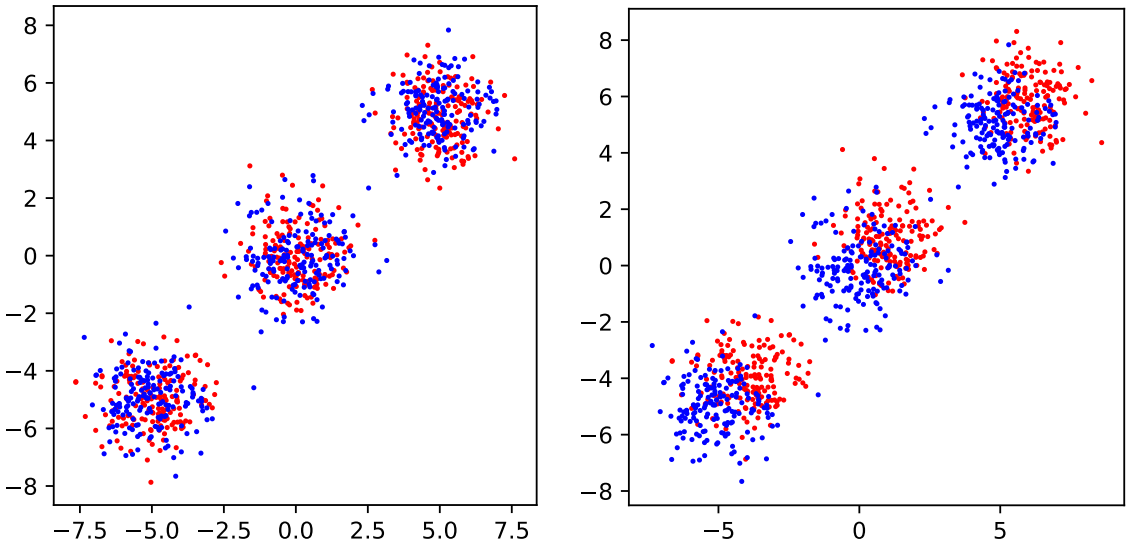

Figure 4.3: Multidimensional KS distance of a 2-dimensional blobs dataset generated with explanatory purposes. Left: the probability distribution is the same in both blue and red samples ($D_{n,m} = 0.062$). Right: the probability distribution is different ($D_{n,m} = 0.177$).

With domain adaptation we are seeking for projections in which data are indistinguishable among domains. If the two-sample KS distance is small when comparing samples from different domains, we could think that the projected data is really domain-independent. The main problem of the KS distance is that it assumes one-dimensional samples. However, there is a generalization of this measure for multidimensional data: the two-dimensional Peacock's distance [36] and its generalization, the multidimensional KS distance [37], which can be applied to compare two-dimensional samples. Figure 4.3 contains an example of how the multidimensional KS distance works. In the first case (left), both the blue sample and the red sample are identically distributed, so we obtain a very small distance (0.062) that indicates that the samples are statistically near. In the second case (right), a constant is added to one of the samples, causing a displacement that prevents the samples from following the same probability distribution. In this case the distance is bigger (0.177), as expected.

Regarding batch normalization, we propose to evaluate the quality of an embedding comparing the multidimensional KS distance before and after the domain adaptation. Before batch correction, we expect the multidimensional KS distance to be large, as samples proceeding from different domains are easily distinguishable. After batch normalization, the distance between domains should be smaller (i.e. we do not expect big differences between the distributions of the samples). These measurements can be used to form a kind of *confusion matrix*, in which rows and columns indicate the domain belonging of data. Making a heatmap, in the case of SAE and t-SNE projections, we expect to see a diagonal matrix, as distances between different domains should be large. However, in the projection learnt by the DAE, we expect to see a block, since, in this case, distances outside the diagonal should be smaller.

5 Experiments

After covering the notions and basics in domain adaptation and adversarial learning, we have introduced the algorithm proposed in this work, the domain-adversarial autoencoder (DAE). In this chapter we present the experiments carried out with flow cytometry data (FCM data) to verify the efficiency of the DAE to perform batch normalization. This chapter is divided in the following sections:

- In Section 5.1 we review the available software resources in Python to analyze flow cytometry data and to perform adversarial learning in neural networks. This section also contains an explanation of the datasets used to test the performance of the algorithm: a synthetic dataset (beads) and real flow cytometry data (dendritic cells).
- In Section 5.2 we explain the experimental set up that we have followed in this work.
- In Section 5.3 we show the DAE's performance on the mentioned datasets, comparing the projections and the clustering results obtained before and after the batch normalization of FCM data.

5.1 Software and datasets

This section is devoted to the available data and software resources used to conduct the experiments of this work. First, we introduce the software and the main Python libraries we have used. Second, we give detailed information about the FCM datasets that have been chosen to test our algorithm, as they had strong batch effects that needed to be corrected for further analysis.

5.1.1 Software

The experiments and code developed in this work are based on standard software libraries. The code has been implemented in Python, using the Keras library to build the neural network architectures. Below, we describe some of the most important libraries that we have used in this work.

Name	N	m	C	l
Beads_4_simple	144701	4	5	5
Beads_4	107702	4	14	2
Dendritic	1843470	11	Unknown	2

Table 5.1: Description of the datasets used in the experiments. N indicates the number of examples, m is the number of channels used for the analysis, C is the number of different populations in the samples (if known) and l is the number of considered domains.

- NumPy [38] is a fast and versatile library whose main data class is an N-dimensional array object. It offers comprehensive mathematical functions to operate on these arrays and it is compatible with Keras and Scikit-learn libraries.
- Keras [39] is a deep-learning API written in Python. It is the high-level API of TensorFlow [40], and provides a convenient interface for solving Machine Learning problems focused on deep learning. This library counts with powerful tools and blocks to efficiency program and test algorithms of Machine Learning, specially neural networks.
- Scikit-learn [41] is an accessible library with efficient tools for data analysis and mining. It contains tools to perform in a very effective way grid searches over the parameters of the models and several dimensionality reduction methods, including implementations of PCA and t-SNE.
- FCSparser (`https://github.com/bpteague/fcsparser`) is a Python package for loading `.fcs` files, which is the most common format in FCM data.
- A Python implementation of the two-dimensional KS distance can be found in `https://github.com/syrte/ndtest`. This code can be used to compare the distribution between two 2-dimensional samples, which is the dimension chosen for all the embeddings computed in this work.
- The implementation of Phenograph used in this work is in R and it is included in the cytofkit package [24], which is an integrated pipeline for mass cytometry analysis.

5.1.2 Datasets

Regarding the experiments carried out in this work, we have chosen three datasets with increasing complexity (two of them are synthetic and one contains real-world data from FCM experiments). Table 5.1 summarizes their main characteristics. Synthetic data are beads stained with a combination of fluorochromes that are passed through the cytometer and analyzed in a very controlled way. Thus, the number of "populations" is known, which makes them very useful data to test the algorithms implemented. Real data consist on mouse cells stained with different combinations of antibodies and fluorochromes, used for state-of-the-art research in immunology, so the actual number of cell populations is unknown.

The first dataset, denoted **beads_4_simple**, contains simple data of beads in a controlled environment. The main characteristics of this dataset are:

- **Number of populations.** There are five pure populations (negatives, AF647+ simple, FITC+ simple, v450+ simple and PE+ simple).
- **Number of channels.** Four conventionally compensated channels: APCREDLASER, FITCBLUELASER, PACIFICBLUEVIOLETLASER and PEYELLOWLASER, which correspond to the markers AF647, FITC, v450 and PE respectively.

- **Batch effect.** Simulation of progressive misaligment of a cytometer mimicked by using different concentrations of fluorochromes when staining the beads of different files (that are acquired sequentially). As result, we have files with the same populations but with differences in the expression levels due to a simulation of misalignment of a cytometer. Altogether, we have five situations (i.e. five domains) with different marker's dilution, where more dilution implies less concentration and less expression levels.

The second dataset, named **beads_4**, is more complicated. In this case, batch effects are not simulated but real, generated by the use of two different machines to acquire the FCM data from equivalent samples. As in the previous dataset, we have simple data of beads that have been acquired in a controlled environment. The main differences are:

- **Number of populations.** There are 14 different classes or populations (combinations of 4 fluorochromes), including negatives examples.
- **Number of channels.** Four conventionally compensated channels: ccV450, ccBV711, ccPE and ccAlexaFluor647.
- **Batch effect.** We have equivalent data acquired in two different machines: FORTESSA (conventional cytometer) and SP6800 (spectral cytometer). Therefore, the first domain is composed by the samples acquired with the conventional cytometer (CONV) and the second domain contains the samples acquired in the spectral cytometer (SPC). Samples were prepared and acquired in the same mode and on the same dates, so the differences between CONV and SPC must be, almost exclusively, due to the use of different cytometers. This is the batch effect we are seeking to correct in our tests.

The third and final dataset, which will bee referred as **dendritic** dataset, comes from a real experiment. It contains equivalent single-cell suspensions of skin-draining lymph nodes from mice acquired in two different machines. Samples of two types of animals are available: WT or control animals, and KO or animals with different genotype (and potentially with some different populations). However, in this work we have used only data from WT animals to constraint the differences between samples to be strictly batch effects instead of a combination with biological changes. On these data, we highlight the following points:

- **Number of populations.** We do not know the total number of cell populations in these samples. We only have a labeled subpopulation by expert immunologists, which is the one of interest in these data, the "migratory dendritic cells", a rare population (i.e. low number of cells) in WT control animals, and even rarer in KO modified animals. As a consequence, classes are now very unbalanced.
- **Number of channels.** Eleven conventionally compensated channels: ccAPC, ccAPC-Fire750, ccBV421, ccBV570, ccBV605, ccBV711, ccFITC, ccPE, ccPE-CF594, ccPE-CY7 and ccPerCP-CY5.5.
- **Batch effect.** We have equivalent data acquired in two different machines: FORTESSA (conventional cytometer, CONV) and SP6800 (spectral cytometer, SPC), as in the beads_4 experiment, so the first domain contains CONV samples and the second domain SPC samples from WT animals. In this case, this is the batch effect we are seeking to correct.

Finally, note that the three datasets have been previously processed by the Cellomics Unit of the Spanish Centro National de Investigaciones Cardiovasculares (CNIC) (see Section 5.2).

5.2 Methodology

In this section we explain thoroughly the procedure followed to perform the experiments and the evaluation strategies to assess the results, which are common to all the tests carried out in this work.

5.2.1 Method design

Regarding the design of the method and the data loading, recall that data have been already pre-processed and normalized with the biexponential transformation (see Section 3.1.1), which has been applied independently for each domain or batch. The first step is to load data from the `.fcs` files through the `fcsparser` library. As mentioned before, target datasets are made up of experiments from different domains. Therefore, we form three objects:

- A data matrix $X \in \mathbb{R}^N \times \mathbb{R}^m$ with the expression levels of each channel (m markers) for each bead or cell (n samples).
- A categorical array Y of dimension N with the information of the population each sample (cell or bead) belongs to (if known). In our case all our experiments are labeled and this data is used for display and evaluation purposes, but this information is not necessary to train the algorithms proposed in this work.
- A categorical array D of dimension N that encodes the domain each sample (cell or bead) belongs to.

We split these objects, finally obtaining a training set $\mathcal{D}_{\text{train}} = \{X_{\text{train}}, Y_{\text{train}}, D_{\text{train}}\}$ and a testing set $\mathcal{D}_{\text{test}} = \{X_{\text{test}}, Y_{\text{test}}, D_{\text{test}}\}$. This partition is done with the `StratifiedKFold` object of Scikit-learn, selecting randomly (in a stratified way) 5/6 of the data for the training subset and 1/6 for the testing subset.

Once the data are correctly loaded and preprocessed, we proceed to train the neural networks. At this point, we study of the distribution of the data and the presence of possible extreme points that may hinder further analysis. Figure 5.1 contains the box-plots for the three datasets and we do not consider the existence of significant outliers in the transformed data that should be removed to train properly the neural networks. Hence, we first train a standard autoencoder (SAE) with the following common parameters, which are common across all datasets.

- Both the encoder and the decoder are composed of 2 hidden layers, each one of 24 units.
- The bottleneck layer is composed of 2 units, as we are interested in 2-dimensional embeddings for display reasons, allowing an appropriate visualization.
- The activation function is the ReLU in hidden layers, linear in the last layer (decoder).
- The learning rate is selected with a grid search over a set of possible values. The space of search is $l_r \in \{1e-05, 1e-04, 1e-03, 1e-02, 1e-01\}$ and the best model is the one obtaining the maximum score. The selected score function is the negative squared root mean error. This is done over 4 training and validation splits with the method `GridSearchCV` available in the library Scikit-learn.
- For the stochastic optimization of the neural networks we use the method Adam, which uses Momentum and adaptive learning rates. It is the optimizer that performs best on average [42].

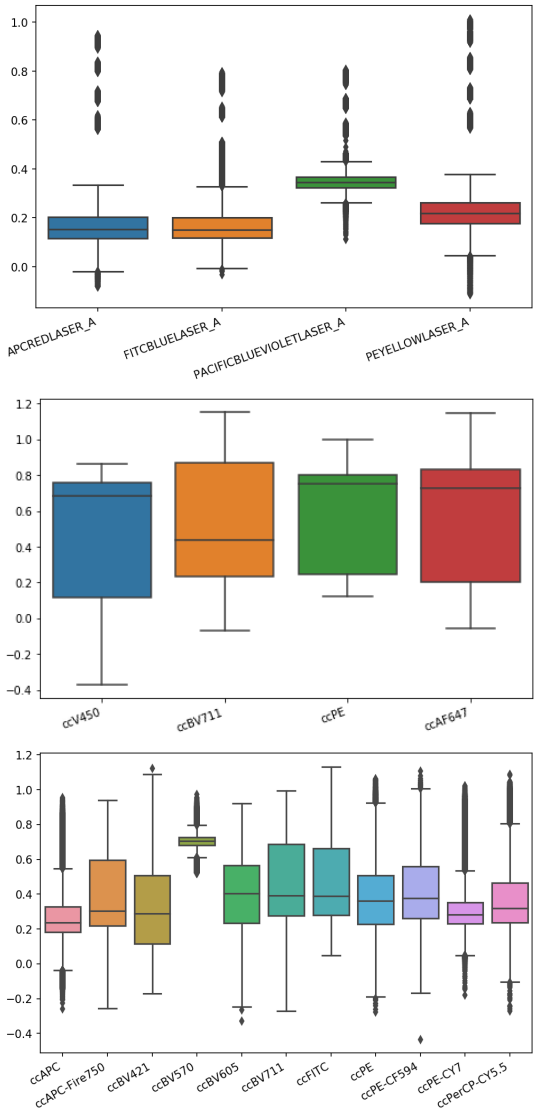

Figure 5.1: Box-plots after applying the logicle transform. Top is the Beads_4_simple, middle is the Beads_4 and bottom is the Dendritic dataset. There are not significant outliers.

- As loss function we use the Mean Squared Error (MSE) between the input and the output.
- We also include regularization in the weights belonging to the encoder, to control the size of the embedding. We use a fixed value $\nu = 10^{-4}$.

Before training the SAE it is necessary to standardize the data. The used transformation is

$$\hat{X} = \frac{X - \mu}{\sigma},$$

where μ is the sample mean and σ is the standard deviation. Both parameters are estimated from the training set, applying the subsequent transformation to both the training and validation sets during the grid search and to the training and testing sets when training the final models. This normalization is also done by batch, which means that the mean and deviation are calculated independently for each domain (or batch). That is, we will have as many tuples (μ_i, σ_i), $0 \leq i < l$, as different experiments, where l denotes the number of domains of the dataset.

Regarding the domain-adversarial autoencoder (DAE), as previously mentioned in Chapter 4, we use a very similar configuration to that of a SAE. We apply the standard normalization per

batch and reuse the optimal parameters of the SAE, such as the learning rate, the architecture and the regularization parameter ν. Concerning the discriminator, we use a net with two hidden layers with 32 units each and with the ReLU as activation functions. The activation of the last layer is the softmax (recall that the discriminator is a net for classification with the task of predicting the domain the data belong to). Thus, the discriminator is trained using the sparse categorical cross-entropy as loss function.

Lastly, for comparison purposes, the t-SNE projections are computed over the testing subset using the standard methods available in Scikit-learn. All parameters are left by default. The same normalization procedure per batch is applied to the data, as algorithms must be compared under the same circumstances.

5.2.2 Evaluation strategy

The evaluation of the results is carried out in three ways. On the one hand, we have the visual evaluation, which is based on verifying that the embeddings appear overlapped by domain or batch in the latent space. On the second hand, on the embeddings computed for the testing subsets, we evaluate the batch normalization with the two-dimensional KS distance and its representation in heatmaps that we have introduced in Chapter 4. When we use a standard autoencoder (SAE) or t-SNE to project data, we do not provide the algorithm with information about batches or domains. Thus, we expect the KS distance to be large, as samples proceeding from different domains are separated in the embedding space, so they are statistically far. However, data from different domains projected with a DAE are expected to be overlapped, so we expect smaller distances between samples. These forms of evaluation are implemented on the testing subsets.

Lastly, we also wanted to evaluate the correction of batch effects in the original space and dimension of the data, and not only in a projected space. To do that, we assess the quality of the reconstruction of the DAE (where batch effects should be removed), compared with the original input data, by computing the $F1$-scores obtained by Phenograph clustering (see Section 3.2) over the data in the original multidimensional space. This is possible because we have labeled data. Note that to evaluate the batch normalization in the reconstruction we need to invert the standard normalization applied before training the networks, which is independent for each domain. As we do not have a source and target domain, because they are all mixed in out partitions, we select the parameters computed for the first domain ($l = 0$), so the reconstruction would be

$$X_{\text{clean}} = \sigma_0 \hat{X} + \mu_0 \ ,$$

where μ_0 and σ_0 are the parameters computed for the first domain of the experiment and X_{clean} denotes the batch normalization of the dataset.

5.3 Results

This section contains the results of batch normalization in the three datasets presented before and with the methodology explained in the previous section. The results of the grid search for the learning rate can be seen in Figure 5.2, where we show both negative squared root mean error (NRMSE) and negative mean absolute error (NMAE). In the case of the beads synthetic datasets, the optimal learning rate is $l_r^* = 10^{-3}$. In the case of dendritic cells, the optimal result is 10^{-4}.

Following the previous methodological scheme, for each dataset firstly we establish a visual evaluation of the results comparing the embeddings obtained by the different methods used in

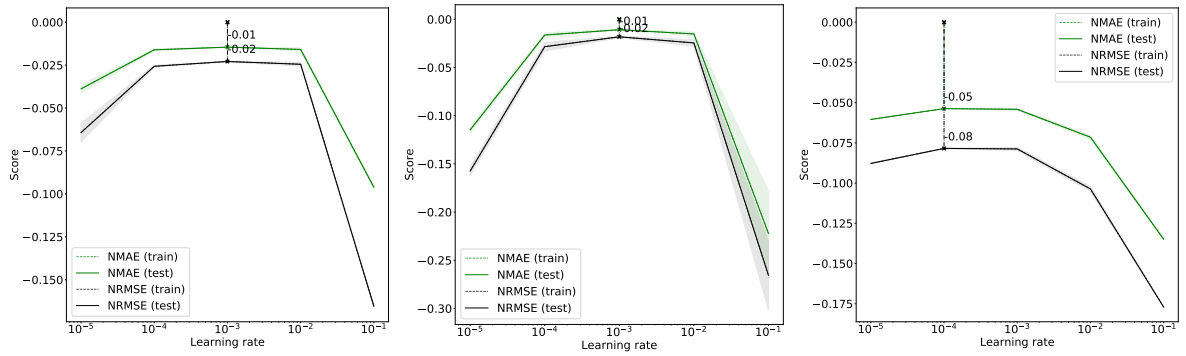

Figure 5.2: Grid search results for the optimal learning rate of a SAE. From left to right: the beads_4_simple, the beads_4 and the dendritic datasets.

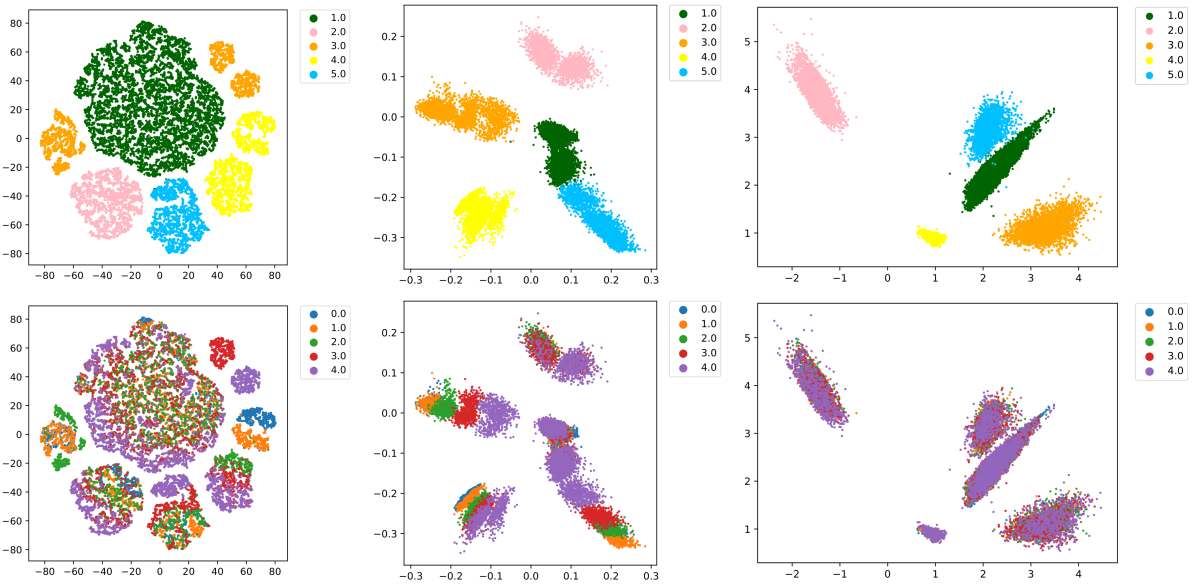

Figure 5.3: Embeddings of the Beads_4_simple dataset. From left to right: t-SNE, SAE and DAE projections. Top row is coloured by class and bottom row is coloured by domain or batch.

this work (t-SNE, SAE and DAE). Secondly, we show the heatmaps containing the multidimensional KS distance before and after batch correction (SAE and DAE, respectively). Finally, we study the efficiency of batch normalization using Phenograph to perform clustering and measuring the $F1$-score in the original and the reconstructed data.

5.3.1 Tests on beads_4_simple dataset

For the Beads_4_simple dataset, we show the t-SNE, SAE and DAE embeddings coloured both by class and domain in Figure 5.3. Clearly, t-SNE and SAE projections are influenced by the domain, as we can see a clear separation in the embedding space. Moreover, in the latent space of the SAE, even though events of the same class tend to come together in the embedded space, the batch effect originated by the decrease in expression values is clearly seen in each domain or batch. These effects are corrected in the embedding computed by the encoder of the DAE, where data is grouped by class or phenotype, but where batch effects are also corrected. Visually, an almost perfect overlap is observed between data from different domains and the same class (or population).

At this point, it seems appropriate to illustrate a little better how the DAE is working. The

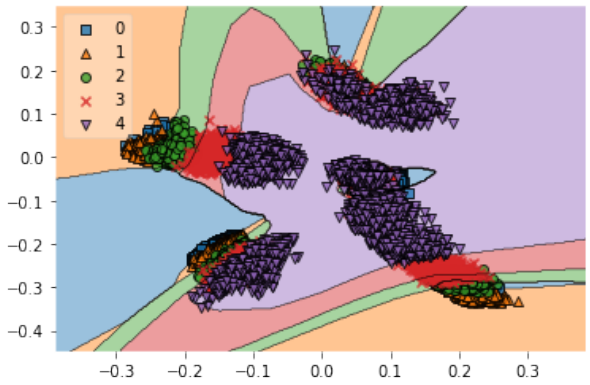

Figure 5.4: Decision regions of a discriminator trained over the SAE projection of the Beads_4_simple dataset consisting on five domains.

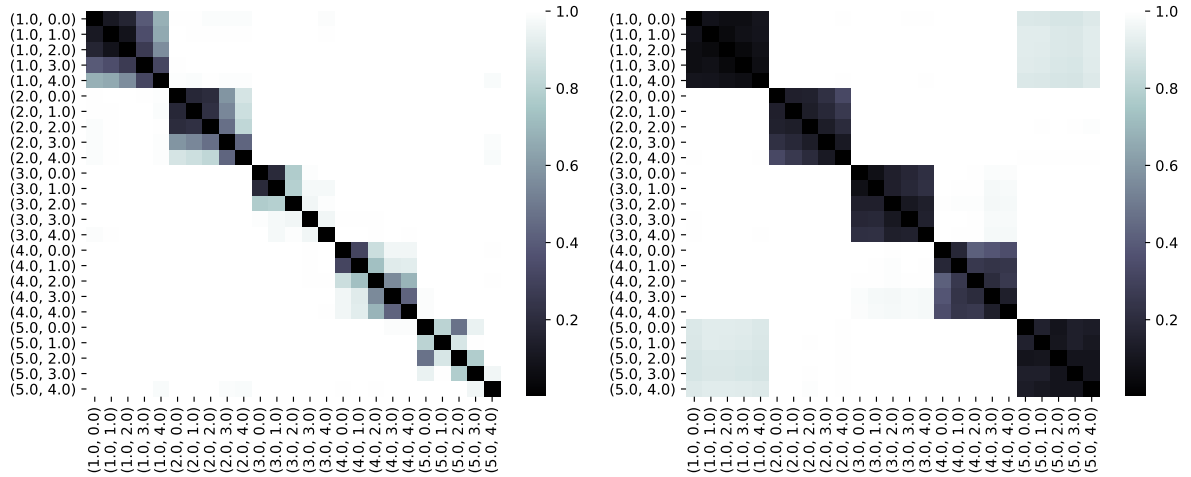

Figure 5.5: Heatmaps with the values of the two-dimensional KS distance on the embeddings of a SAE (left) and a DAE (right) in the Beads_4_simple dataset. Tuples indicate (class, domain).

encoder of the DAE tries to fool the domain discriminator, and the way of achieving this is to superimpose data in the embeddings so that the domains are not distinguishable. For example, in Figure 5.4 we can see the decision regions of a neural network for classification with the same architecture and parameters used in the discriminator. The neural network is trained with the SAE projection and it clearly learns to distinguish data from different domains. As the encoder is trained to fool a network of the same type, it learns to project data into a latent space where data is not as clearly separable (by domain but by class) as long as the decoder reconstruction error is still acceptable.

These results are supported by the heatmaps containing the two-dimensional KS distance, which can be seen in Figure 5.5. In the image on the left, we see that data from the same domain present large distances, which means that the empirical probability distributions of data from the same class but different domains are different. However, in the DAE projection shown on the right, we see well-defined blocks. While the distributions of the classes do not mix, we see how the distances between projections of data from different domain decrease with respect to previous values. This is precisely the effect sought in this work.

Finally, we make some tests over the DAE reconstructions, in which we expect a considerable batch correction that results in improved Phenograph clustering results. Before batch correction, the clustering results can be seen in the left part of Table 5.2. We can see that, in general,

CLASS	DOMAIN									
	Before					After				
	0	**1**	**2**	**3**	**4**	**0**	**1**	**2**	**3**	**4**
1	0.319	0.34	0.369	0.354	0.385	0.444	0.473	0.477	0.482	0.465
2	1	1	0	0	0	0.676	0.658	0.609	0.621	0.588
3	1	1	0	0	0	0.521	0.478	0.439	0.4	0.363
4	1	1	0	0	0	0.813	0.759	0.629	0.705	0.619
5	1	1	0	0	0	0.66	0.526	0.561	0.558	0.515
ALL	0.737	0.709	0.205	0.182	0.199	0.571	0.546	0.522	0.536	0.499

Table 5.2: $F1$-scores before and after batch normalization obtained via automatic clustering with Phenograph in the beads_4_simple dataset. Global before: 0.439. Global after: 0.534

each population (i.e. each class) of each of the domains is detected as a different cluster; and even for some of the clusters we obtain a perfect $F1$-score. However, some groups are totally missclassified, resulting in a low global $F1$-score. The results after batch normalization can be seen in the right columns of Table 5.2. As we can see, there are no longer clusters that correspond to populations of only one of the domains or batches, it is instead a mixture of all. However, since there is a lot of overclustering in each population (i.e. class), the $F1$-scores are lower than expected. Anyways, it improves the pre-correction value globally. The differences shown in the table can be summarize as follows:

- Pre-correction: a chosen cluster can either match a class perfectly for some domains (e.g. domains 0 and 1 achieve perfect $F1$-score for class 5), but without containing any event of the same class belonging to other domains (domains 2 to 4 for the same class 5).
- Post-correction: a chosen cluster has events from all domains (since they are no longer separated), but there is overclustering.

Therefore, for the beads_4_simple dataset, the results of the batch correction are very positive, since we can see an improvement of the display of the data in the embedding, where the domains overlap but the different classes remain separated; in the evaluations made with the two-dimensional KS distance on the latent space, obtaining smaller distances between batches or domains in the DAE projection; and in the $F1$-score obtained by Phenograph using the reconstructed data obtained by the DAE.

5.3.2 Tests on beads_4 dataset

The beads_4 dataset, as mentioned in the first section of the chapter, is composed of data coming from two different machines: the conventional (CONV) and spectral (SPC) cytometers. Therefore, this time we consider two domains or batches that we refer to as CONV and SPC. The t-SNE, SAE and DAE projections of these data can be seen in Figure 5.6. Regarding the t-SNE projection, we can see that test examples are completely separated by domains, thus, appearing in the latent space twice the number populations we know there are in the sample (recall this synthetic dataset is labeled). The SAE projection is also sensible to batch effects, even though these effects are visually soften in comparison to the t-SNE embedding. This means that the positioning of events is more realistic, as events or cells that are supposed to be similar are nearer in the embedding space (e.g. same class for different domains). Clearly, the DAE projection presents the highest visual degree of overlap between domains, while events remain class-separated.

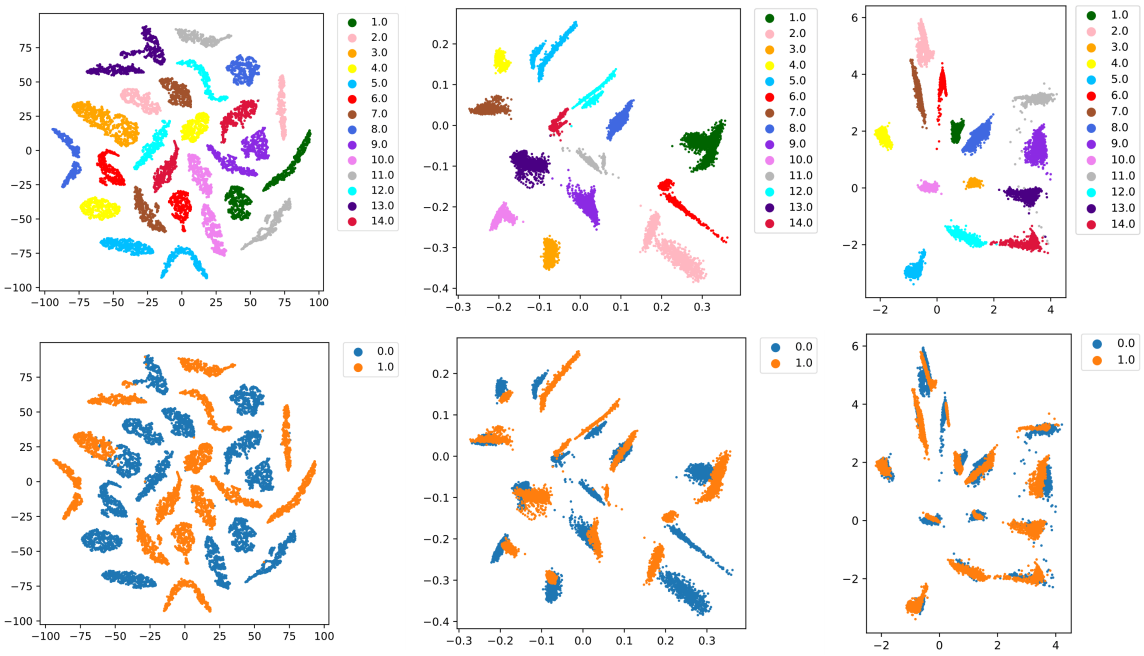

Figure 5.6: Embeddings of the beads_4 dataset. From left to right: t-SNE, SAE and DAE projections. Top row is coloured by class and bottom row is coloured by domain or batch.

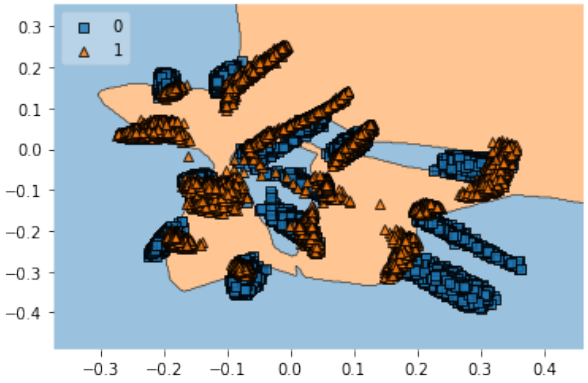

Figure 5.7: Decision regions of a discriminator trained over the SAE projection of the beads_4 dataset.

We also show the decision regions obtained after training 10 epochs a neural network for classification, with the same configuration of the discriminator, on the embedding learned by the SAE (see Figure 5.7). As there are domain differences in the latent space, the network achieves a 93% of accuracy. However, if we train for the same number of epochs the same net, but this time on the embedding learned by the DAE, the classification accuracy decays to a 68.19%, so for the network it is harder to distinguish between domains. In the same way, we appreciate the overlapping in the visual evaluation. Be aware that the baseline accuracy is approximately 50%, as in this dataset, there are two classes that are balanced between domains; so a trivial classifier would guess correctly half of the examples.

To support these results, we show the heatmaps with values of the two-dimensional KS distance in Figure 5.8. This time, despite the clear improvement obtained in the DAE projection, there are still some differences in the way that data from different domains are distributed. For example, events of class 9, although they are very close, do not present the same mean given the domain. Either way, given the same source data, the distance associated with the DAE

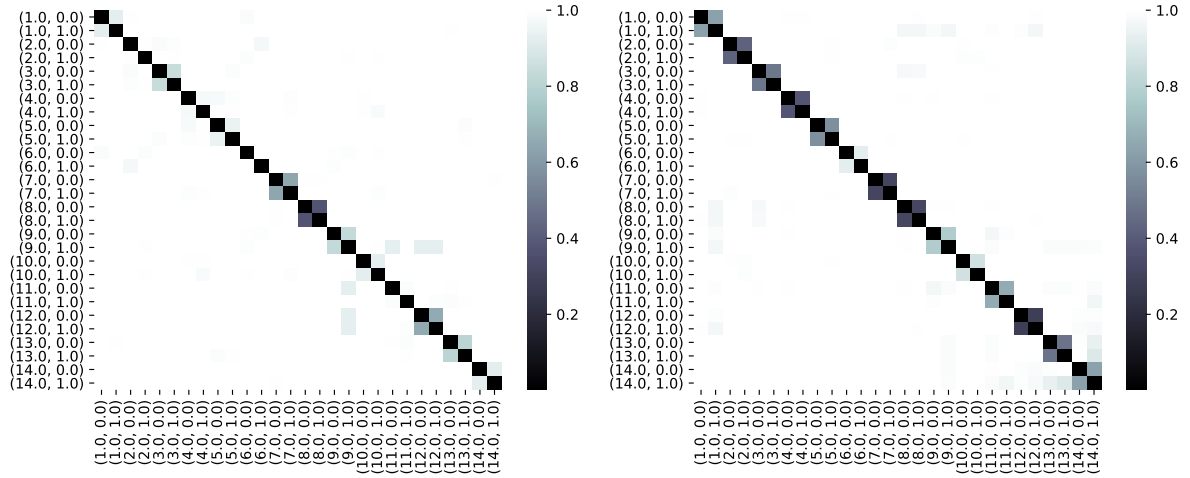

Figure 5.8: Heatmaps with the values of the two-dimensional KS distance on the embeddings of a SAE (left) and a DAE (right) in the beads_4 dataset. Tuples indicate (class, domain).

	DOMAIN			
CLASS	**Before**		**After**	
	CONV	**SPC**	**CONV**	**SPC**
1	1	0	0.790	0.718
2	0.801	0	0.890	1
3	1	1	0.999	1
4	1	0	1	1
5	1	1	0.66	1
6	1	0	0.999	1
7	1	0.001	1	1
8	1	0	0.650	0.789
9	0.631	1	0.946	0.810
10	1	1	0.999	1
11	0.993	0.001	0.999	0.999
12	0.998	0.998	0.862	0.774
13	1	0.999	0.996	0.809
14	0.995	0.998	0.968	0.970
ALL	0.968	0.665	0.943	0.928

Table 5.3: $F1$-scores before and after batch normalization obtained via automatic clustering with Phenograph in the beads_4 dataset. Global before: 0.841. Global after: 0.936.

projection is always smaller than the distance computed with the SAE projection. Therefore, in this particular dataset, the DAE is bringing closer the distributions of the projected data among domains or batches, as expected.

For these data, clustering results are especially good. We can see in Table 5.3 a very significant improvement in the results. Especially for the domain SPC, which had very low $F1$-scores in some classes before applying batch normalization. This is due to the appearance of these artificial groups we observed in the SAE projection of Figure 5.6, where we can see that some classes appear separated in the embedding space because of the batch effects. The specific $F1$-score for the SPC domain changes from 0.665 in the original data to 0.928 in the corrected data. The global $F1$-score is 0.841 for non-corrected data, while we obtain 0.936 in the data corrected by the domain-adversarial autoencoder.

Taking this into account, we can conclude that we have satisfactorily removed a large majority of the batch effects present in this dataset. In favour of the relevance of this result we must advert that the batch effects were originated from using two different machines to acquire the FCM data, which is an extreme situation that does not normally occur in flow cytometry laboratory approaches. Thus, one can consider this batch effect an extreme perturbation of the experimental conditions, which goes beyond any experimental condition commonly used in flow cytometry real experiments. The effects of the correction are visually observed in the embeddings and objectively measured in the clustering results.

5.3.3 Tests on dendritic dataset

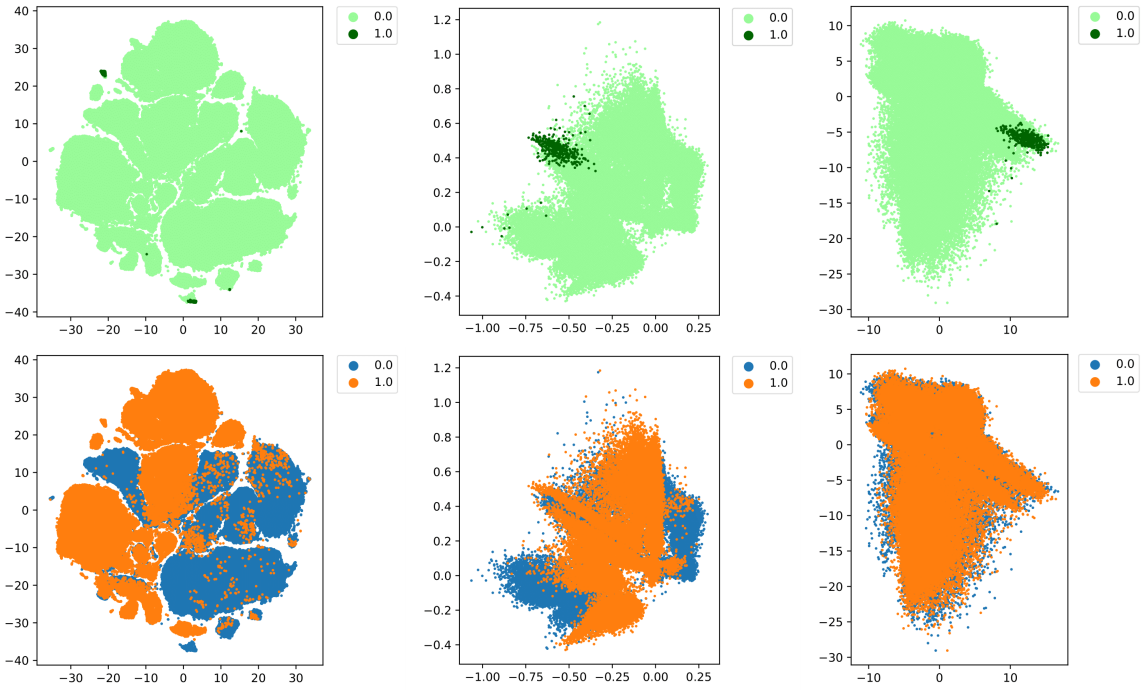

Figure 5.9: Embeddings of the dendritic dataset. From left to right: t-SNE, SAE and DAE projections. Top row is coloured by class and bottom row is coloured by domain or batch.

Finally, in this section, we present the results obtained in a real-world experiment. This dataset contains dendritic cells of mice (see Section 5.1.2 for more details) that have been acquired using two different machines. Again, an extreme perturbation is introduced as a batch effect, which is not a common task to encounter in flow cytometry approaches. As in the previous experiment, we consider two domains: CONV, which are the equivalent samples acquired in the conventional cytometer; and SPC, which are the samples acquired in the spectral cytometer. Recall that we have only one population labeled in this dataset, which is the population of biological interest, and we are using only data from WT animals, so we are sure that differences between domains are strictly batch effects and not biological changes. In addition, this population is considered a "rare population" in cytometry; that is, it constitutes a very small percentage of cells considering the whole dataset, and, therefore, it is difficult to detect. Actually, we are facing a dataset with few labels, which could be considered in a semi-supervised learning context.

Figure 5.9 contains the t-SNE, SAE and DAE projections of this dataset. The batch effects have a strong influence in the t-SNE and SAE embeddings. Continuing with the results obtained in the tests with synthetic beads, the DAE manages to project the data in a space where the

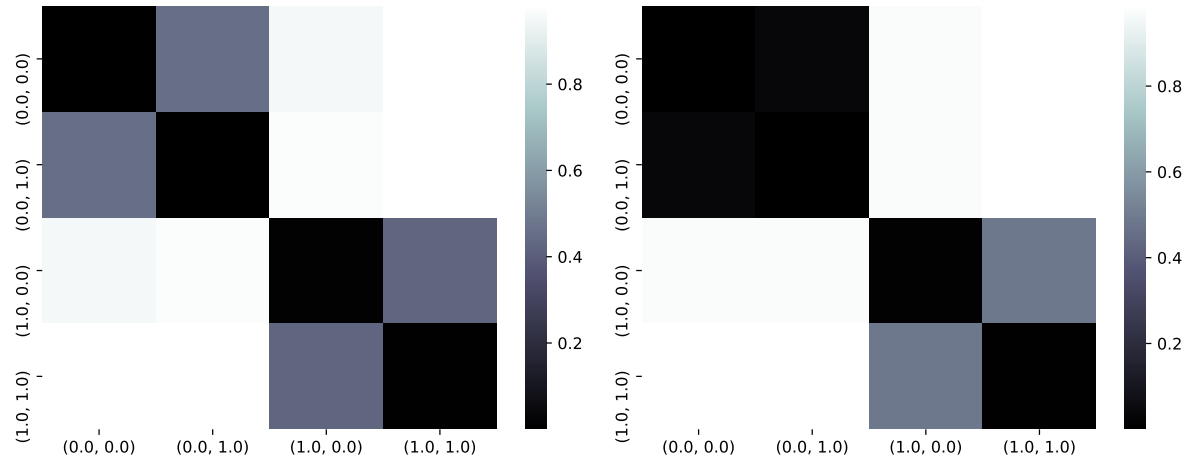

Figure 5.10: Heatmaps the values of the two-dimensional KS distance on the embeddings of a SAE (left) and a DAE (right) in the dendritic dataset. Tuples indicate (class, domain).

differences among domains are not significantly appreciated in the embedding. The improvement is visually evident: events that were hardly in contact in the t-SNE and SAE projections, now have a very high degree of overlapping.

This fact can be verified in the heatmaps of the two-dimensional KS distance, which can be seen in Figure 5.10. Concerning the SAE, we can see that there are already similarities in the populations. Clearly, this is because the events of the target and other unknown cell populations fall very close in both domains in the projection, which indicates that the autoencoder is already bringing closer events that are biologically similar, independently on the batch or experiment. Although the distributions are still clearly distinguishable. With respect to the DAE projection, we can see an obvious improvement, as the target population still shares a small distance between domains; and the events with label 0, which are the vast majority, present almost distance 0 when comparing samples acquired in different machines. Therefore, again, the distribution of the projected data between domains or batches is much more similar in the case of the DAE, indicating a clear adaptation to the domain also in data from real cells.

CLASS	**DOMAIN**			
	Before		**After**	
	CONV	**SPC**	**CONV**	**SPC**
1	0.002	0.206	0.107	0.247
ALL	0.002	0.206	0.107	0.247

Table 5.4: $F1$-scores before and after batch normalization obtained via automatic clustering with Phenograph in the dendritic dataset. Global before: 0.195. Global after: 0.173.

Finally, we can compare the F1-score obtained in the clustering with Phenograph (Table 5.4). This time, the score is only computed for the labeled class. As the tagged target population is not very far between domains initially (before correction), we do not start with much room for improvement in the numerical results. Even though, contrary to what happens with the two-dimensional KS distance, the $F1$-score does not result optimal, going in line with the projections observed in Figure 5.9, we do consider this a positive result; as before correction, there are failures in the clustering because part of the tagged population from one of the domains falls into different clusters (Phenograph is able to find events from one domain but not from the other). However, after correction, although the selected cluster guesses correctly events from both domains; it is also a larger cluster with events belonging to other populations that are not

identified. All of this, together with the fact that our reference standard is a rare population and there is a noticeable unbalance of cells from this class between the two domains, make the global $F1$-score similar before and after the correction (slightly lower, in fact). Anyway, the individual $F1$-scores for each file/domain are improved after the batch correction with the DAE. In any case, the proper detection and clustering of this population of migratory dendritic cells, have been demonstrated to be extremely difficult [2], even when the full spectra for expression levels are available when acquiring the data with the cytometer (which is not our case, since we are using classical conventional channels). Nevertheless, the KS distance gives us room for expect an improvement of the $F1$-scores in more realistic batch effects, which remain to be tested.

Therefore, despite these clustering results in a complex dataset that has a huge deficit in population labeling, we can conclude that the batch correction has been satisfactory. Both the visual results and the statistical distance measurements show a clear attenuation of the differences between the different domains, which, due to the characteristics of the dataset used, correspond almost entirely to batch purposes.

6

Discussion and further work

The problem of batch normalization in biological experiments is important to adequately introduce automated pipelines of analysis that study samples that may have been acquired in different experimental conditions, which may introduce non-biological variations that can complicate further studies. In this work, we have applied state-of-the-art Machine Learning disciplines, such as domain adaptation and adversarial learning with deep neural networks, to mitigate the batch effects in flow cytometry (FCM) data. We have carried out diverse experiments with three types of data: synthetic data with simulated batch effects, synthetic data with extreme batch effects and real data from dendritic cells with extreme batch effects.

Initially, we proposed a method of domain adaptation since it possesses several features that would fit theoretically well with the task at hand. First, this work proposes a new method for domain adaptation in the context of unsupervised learning: the domain-adversarial autoencoder (DAE). Second, in the particular case of FCM data, we have considered that data acquired in different conditions (batches) defined different domains in our experiments. In the experimental approaches undertaken in this work we were able to ascertain that the benefit of applying domain-adversarial autoencoders is double. On the one hand, we obtain corrected data as output of the decoder. On the other hand, we obtain a representation in a space of smaller dimension, almost free of batch effects. For checking these results, we have carried out two types of experiments: we have evaluated the corrected reconstructed data using Phenograph, a clustering algorithm, proving that the new method achieves better clustering results; and we have also compared the embeddings obtained via t-SNE, a standard autoencoder (SAE) and a DAE, assessing the removal of the batch effects of the last one with respect to the first two techniques. The embedding comparison has been done both visually and numerically, using a statistical distance that compares the distribution of data over domains in the latent spaces of SAE and DAE.

The experimental approaches undertaken for this project allowed us to verify that batch effects greatly influence the outcome of automated pipelines of analysis of FCM data. For example, we have seen that when using t-SNE to project the data into a 2-dimensional space, the data is mainly distributed according to the origin of the experiment, instead of its phenotype. We have also observed the same effects applying clustering techniques: obtaining almost a perfect grouping in some of the domains or batches, and null results in others. The results presented in the memory (Chapter 5) enable us to draw important conclusions:

- Regarding the visual results, we can see, as expected, a clear visual separation by batches or domains in the projections calculated by t-SNE and the SAE in all datasets. Importantly, we gain a great improvement by the projections obtained by DAE, since we can see a manifest overlap in correspondent populations, being the visual distinction between domains much more difficult in this second case, but still maintaining the phenotipic differences between populations. This has important applications in the flow cytometry field, since most of the present data representations rely on these projections, and will greatly benefit from the method developed herein.

- Regarding the evaluation of the embeddings with the multidimensional KS distance, we have also obtained very promising results. In all the experiments carried out, we have systematically attained smaller values of distances with the DAE than with the SAE. This measure is a first approximation to a possible objective measurement of the efficiency of batch effects' removal obtained by the method developed within this work.

- The results of the $F1$-score in the clustering performed by Phenograph also seem very promising, albeit we obtained better results with the synthetic datasets than with the real biological samples, as expected. In the dendritic dataset, the results are less optimistic because we have very few tagged events of the class of interest, and, although a clear visual overlap exists in the target class in the DAE embedding, which makes it a promising approach to pursue in future investigations, the global $F1$-score slightly decreases after the correction. Remember that these scores are calculated on the original complete datasets (before) and with the same data at the output of the decoder (after). In addition, the batch effects target of corrections were extreme cases, which opens up some room for improvement when more realistic batch effects are tested.

Therefore, the domain-adversarial autoencoder has been validated by our experiments as an efficient algorithm to palliate batch effects in FCM data. The developed method allows working with multiple domains or batches (two or more) simultaneously, which is a huge advantage with respect to other similar approaches in the literature. Visually, we obtain reliable projections in 2-dimensional spaces, which are not as separated by batch or experiment as before, providing the user with a very powerful tool to visualize and understand their data, which opens up new opportunities to improve the state of the art in flow cytometry data analysis; as stated below. This visual correction is objectively supported by the results obtained with the multidimensional KS distance in all our experiments; and also the clustering results with Phenograph are better after the batch correction performed by the DAE, effectively reducing the influence of the batch effects in both synthetic datasets. In addition, with the DAE, we obtain both corrected data and an alternative dimensionality reduction to t-SNE, avoiding many of its problems: the computational cost, the inability to project new data and the stochastic component.

The research work presented herein has still much room for improvement if further work is undertaken following the lines stated below:

- For visualization reasons and as a first approximation, this work is focused on calculating projections or embeddings of the data in 2-dimensional spaces, and we have restricted ourselves to the channels typically analyzed by flow cytometry users, despite having a broader spectrum available. However, the dimension of the bottleneck layer can be enlarged, just as more channels can be introduced into the analysis, possibly improving the quality of the reconstruction.

- The assessment of the batch correction is still an open problem. In this work we give a possible measure for its evaluation, but many others statistical distances could be used instead. As future work we leave the review of the existing literature in this field, as well as the implementation and testing of other notions of distance.

- The purpose of this work is to alleviate the batch effects in samples that have been acquired in different conditions but with the same biological information. That is, the DAE acts as a batch effect remover for these samples, offering a representation without batch information. However, it is also interesting to project and reconstruct out-of-sample data, with possibly different characteristics, using the DAE. For example, we could use data from two different genotypes: cells from wild type (WT) mice, and cells from some knock-out (KO) mice, which contain, in principle, similar but potentially different populations that are beyond our knowledge. In this scenario, we could train the DAE with WT data, and apply the batch correction to KO data.

- This work is restricted to the analysis of flow cytometry data. However, the applications of the domain-adversarial autoencoder can be extended to any unsupervised biological problem in which batch effects complicate the integration of data from different experiments. Moreover, it can be extended to general machine learning problems that are composed of data from different but similar distributions in an unsupervised manner. One of the main objectives is to extend the tests performed in this work to fully validate the usage of the domain-adversarial autoencoder for domain adaptation in unsupervised learning.

Glossary of acronyms

- **SAE**: standard autoencoder.
- **DAE**: domain-adversarial autoencoder.
- **OOS**: out-of-sample.
- **FCM**: flow cytometry.
- **KS**: Kolmogorov-Smirnov.
- **WT**: wild-type.
- **KO**: knock-out.

Bibliography

[1] Matthew H Spitzer and Garry P Nolan. Mass cytometry: single cells, many features. *Cell*, 165(4):780–791, 2016.

[2] Daniel Jimenez-Carretero, José M Ligos, María Martínez-López, David Sancho, and María C Montoya. Flow cytometry data preparation guidelines for improved automated phenotypic analysis. *The Journal of Immunology*, 200(10):3319–3331, 2018.

[3] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[4] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.

[5] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[6] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[7] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[8] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Advances in neural information processing systems*, pages 137–144, 2007.

[9] Anindya Gupta, Philip J Harrison, Håkan Wieslander, Nicolas Pielawski, Kimmo Kartasalo, Gabriele Partel, Leslie Solorzano, Amit Suveer, Anna H Klemm, Ola Spjuth, et al. Deep learning in image cytometry: a review. *Cytometry Part A*, 95(4):366–380, 2019.

[10] Hanling Wang, Mingyang Li, Fei Ma, Shao-Lun Huang, and Lin Zhang. Unsupervised anomaly detection via generative adversarial networks. In *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 313–314. IEEE, 2019.

[11] Matthew Amodio, David Van Dijk, Krishnan Srinivasan, William S Chen, Hussein Mohsen, Kevin R Moon, Allison Campbell, Yujiao Zhao, Xiaomei Wang, Manjunatha Venkataswamy, et al. Exploring single-cell data with deep multitasking neural networks. *Nature methods*, pages 1–7, 2019.

[12] Uri Shaham, Kelly P Stanton, Jun Zhao, Huamin Li, Khadir Raddassi, Ruth Montgomery, and Yuval Kluger. Removal of batch effects using distribution-matching residual networks. *Bioinformatics*, 33(16):2539–2546, 2017.

[13] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[14] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[15] David Pollard. Strong consistency of k-means clustering. *The Annals of Statistics*, pages 135–140, 1981.

[16] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In *International Workshop on Algorithms and Computation*, pages 274–285. Springer, 2009.

[17] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[18] Quoc V Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *ICML*, 2011.

[19] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[20] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[22] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[23] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[24] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. *arXiv preprint arXiv:1206.4683*, 2012.

[25] Aysun Adan, Günel Alizada, Yağmur Kiraz, Yusuf Baran, and Ayten Nalbant. Flow cytometry: basic principles and applications. *Critical reviews in biotechnology*, 37(2):163–176, 2017.

[26] Yvan Saeys, Sofie Van Gassen, and Bart N Lambrecht. Computational flow cytometry: helping to make sense of high-dimensional immunology data. *Nature Reviews Immunology*, 16(7):449–462, 2016.

[27] David R Parks, Mario Roederer, and Wayne A Moore. A new "logicle" display method avoids deceptive effects of logarithmic scaling for low signals and compensated data. *Cytometry Part A: The Journal of the International Society for Analytical Cytology*, 69(6):541–551, 2006.

[28] Jacob H. Levine, Erin F. Simonds, Sean C. Bendall, Kara L. Davis, El ad D. Amir, Michelle D. Tadmor, Oren Litvin, Harris G. Fienberg, Astraea Jager, Eli R. Zunder, Rachel Finck, Amanda L. Gedman, Ina Radtke, James R. Downing, Dana Pe'er, and Garry P. Nolan. Data-driven phenotypic dissection of AML reveals progenitor-like cells that correlate with prognosis. *Cell*, 162(1):184–197, jul 2015.

[29] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[30] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2018.

[31] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[32] El-ad David Amir, Kara L Davis, Michelle D Tadmor, Erin F Simonds, Jacob H Levine, Sean C Bendall, Daniel K Shenfeld, Smita Krishnaswamy, Garry P Nolan, and Dana Pe'er. visne enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia. *Nature biotechnology*, 31(6):545–552, 2013.

[33] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *Advances in neural information processing systems*, pages 513–520, 2007.

[34] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[35] Howard G Tucker. A generalization of the glivenko-cantelli theorem. *The Annals of Mathematical Statistics*, 30(3):828–830, 1959.

[36] John A Peacock. Two-dimensional goodness-of-fit testing in astronomy. *Monthly Notices of the Royal Astronomical Society*, 202(3):615–627, 1983.

[37] Giovanni Fasano and Alberto Franceschini. A multidimensional version of the kolmogorov–smirnov test. *Monthly Notices of the Royal Astronomical Society*, 225(1):155–170, 1987.

[38] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[39] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[40] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[41] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.