EPFL

Thèse n° 8044

Graph Representation Learning with Optimal Transport: Analysis and Applications

Présentée le 25 mai 2022

Faculté des sciences et techniques de l'ingénieur
Laboratoire de traitement des signaux 4
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Effrosyni SIMOU

Acceptée sur proposition du jury

Prof. J.-Ph. Thiran, président du jury
Prof. P. Frossard, directeur de thèse
Prof. X. Bresson, rapporteur
Dr J. Feydy, rapporteur
Prof. P. Vandergheynst, rapporteur

École polytechnique fédérale de Lausanne

2022

Love and respect what you do.
— Sylvie Guillem

To my family...

Acknowledgements

The journey of my PhD studies has been very rich and full of experiences which led me to evolve academically and personally. I therefore want to take a moment to thank all the people that contributed to these experiences.

First, I have to thank my supervisor Prof. Pascal Frossard, for giving me the opportunity to join his research group and for trusting me with this PhD thesis. I also thank him for encouraging me to chose my research topic, which I truly enjoyed to work on.

Further, I would like to thank the members of my thesis committee Prof. Vandergheynst, Prof. Bresson and Dr. Feydy for the time they took to review my thesis and provide valuable suggestions and feedbacks. Their interesting questions made my PhD exam a far more pleasant experience than I could have ever imagined it to be. I also want to say a big thank you to the jury president Prof. Thiran for his help with the organizational aspects of my thesis defence.

A special thank you goes out to Dorina Thanou, with whom I had the opportunity to collaborate on a paper. Dorina's genuine belief in me and my work gave me a significant confidence boost, when I most needed it. I thank her for taking the time to give me useful feedback, for helping me organize my ideas and for motivating me.

I want to thank also all the members of the LTS4 lab and, in general, the LTS corridor who made me happy to go to EPFL everyday. I thank Renata for her warm personality, for sharing our birthday celebration and for encouraging me to continue my ballet classes when I arrived in Lausanne. I thank Isabela for being such a cool, fun person and for the lovely time we had in Brighton together. A big thank you to Eda for the interesting discussions on graphs, books, music.., for our long walks in the Ecublens nature during the Covid quarantine and for always being there to share good advice. I also thank Marwa for her positive vibes; I certainly missed her big smile since she graduated. Thank you also to Clémentine for her help with the French version of the abstract of this thesis.

I cannot think of my years in Lausanne without thinking of my roommate and friend Anne-Sophie. I thank her from the bottom of my heart for being such a kind, warm-hearted person and for filling our house with laughter, cooking and Jasmine's barking. I have the best memories of our living together.

A big thank you also to all of my friends from Greece, who are by my side even though we are currently scattered all over the globe. A special thank you to my friend Anthi, for her incredible ability to empathize, for always telling me exactly what I need to hear and for inspiring me to think out of the box.

Acknowledgements

A huge thank you to Giorgos for being with me during all the years of my master's and PhD studies. I thank him for being a constant source of love and balance in my life and for supporting me in pursuing my goals. I also thank him for learning German and for coming to join me in Switzerland when I started my PhD.
Last, but certainly not least, I thank my parents and my sister Eva for their infinite love and support. I thank them for believing in me and for always giving me a reason to be happy. I literally cannot find the words to express how grateful I am to have them in my life. I dedicate this thesis to them.

Zürich, November 15, 2021

Abstract

In several machine learning settings, the data of interest are well described by graphs. Examples include data pertaining to transportation networks or social networks. Further, biological data, such as proteins or molecules, lend themselves well to graph-structured descriptions. In such settings, it is desirable to design representation learning algorithms that can take into account the information captured by the underlying structure. This thesis focuses on providing new methods to ingrain geometrical information in end-to-end deep learning architectures for graphs through the use of elements from Optimal Transport theory.
First, we consider the setting where the data can be described by a fixed graph. In this setting, each datapoint corresponds to a node of the graph and the edges among nodes signify their pairwise relations. We propose an autoencoder architecture that consists of a linear layer in the encoder and a novel Wasserstein barycentric layer at the decoder. Our proposed barycentric layer takes into account the underlying geometry through the diffusion distance of the graph and provides a way to obtain structure-aware, non-linear interpolations, which lead to directly interpretable node embeddings that are stable to perturbations of the graph structure. Further, the embeddings obtained with our proposed autoencoder achieve competitive or superior results compared to state-of-the-art methods in node classification tasks.
Second, we consider the setting where the data consist of multiple graphs. In that setting, the graphs can be of varying size and, therefore, a global pooling operation is needed in order to obtain representations of fixed size and enable end-to-end learning with deep networks. We propose a global pooling operation that optimally preserves the statistical properties of the representations. In order to do so, we take into account the geometry of the representation space and introduce a global pooling layer that minimizes the Wasserstein distance between a graph representation and its pooled counterpart, of fixed size. This is achieved by performing a Wasserstein gradient flow with respect to the pooled representation. Our proposed method demonstrates promising results in the task of graph classification.
Overall, in this thesis we provide new methods for incorporating geometrical information in end-to-end deep learning architectures for graph structured data. We believe that our proposals will contribute to the development of algorithms that learn meaningful representations and that take fully into account the geometry of the data under consideration.
Keywords: graph representation learning, optimal transport

Résumé

Dans plusieurs contextes liés à l'apprentissage automatique, les données d'intérêt peuvent être décrites par des graphes. Il s'agit par exemple de données relatives aux réseaux de transport ou aux réseaux sociaux. De plus, les données biologiques, telles que les protéines ou les molécules, se prêtent bien à des représentations graphiques. Dans de telles situations, il est souhaitable de concevoir des algorithmes d'apprentissage de représentations qui prennent en compte l'information venant de la structure sous-jacente des données. Cette thèse se concentre sur de nouvelles méthodes afin d'incorporer l'information géométrique dans des architectures d'apprentissage complet et profond pour les graphes par l'usage d'éléments provenant de la théorie du transport optimal.

Tout d'abord, nous considérons une situation dans laquelle les données peuvent être décrites par un graphe fixe. Dans ce cas, chaque point de données correspond à un nœud du graphe et les arêtes entre les nœuds représentent leurs relations par paire. Nous proposons une architecture d'un auto-encodeur qui se compose d'une couche linéaire pour l'encodeur et d'une nouvelle couche Wasserstein barycentrique pour le décodeur. Notre couche barycentrique proposée prend en compte la géométrie sous-jacente en utilisant la distance de diffusion du graphe et permet d'obtenir des interpolations non-linéaires et conscientes de la structure, qui conduisent à des intégrations de nœuds directement interprétables et résistantes aux perturbations de la structure du graphe. De plus, les intégrations obtenues par l'auto-encodeur proposé ont atteint des résultats compétitifs ou supérieurs aux méthodes proposées dans l'état de l'art pour des tâches de classification de nœuds.

Dans un second temps, nous considérons une situation où les données sont décrites par plusieurs graphes. Dans ce cas, les graphes peuvent être de taille variable et, par conséquent, une opération de regroupement globale est nécessaire afin d'obtenir de représentations de taille fixe et de permettre un apprentissage complet avec des réseaux profonds. Nous proposons une opération de regroupement global qui préserve de manière optimale les propriétés statistiques des représentations. Pour ce faire, nous prenons en compte la géométrie de l'espace des représentations et introduisons un regroupement global qui minimise la distance de Wasserstein entre une représentation graphique et son homologue regroupé, de taille fixe. Ceci est réalisé grâce à un flux de gradient de Wasserstein par rapport à la représentation regroupée. La méthode proposée montre des résultats prometteurs dans les tâches de classification de graphes.

Dans l'ensemble de cette thèse, nous proposons de nouvelles méthodes pour incorporer

l'information géométrique dans des architectures d'apprentissage complet et profond pour des données structurées en graphes. Nous pensons que nos propositions contribueront au développement d'algorithmes d'apprentissage de représentations graphiques qui apprennent des représentations significatives et qui prennent en compte la géométrie des données considérées.

Mots clés : apprentissage des représentations pour les graphes, transport optimal

List of Tables

List of Figures

Contents

1 Introduction

1.1 Motivation

In many machine learning settings the data of interest can be described using graph structures. Such settings arise, for instance, in the analysis of the behaviours of users in social networks, in the communication of nodes in sensor networks and in molecule generation for drug discovery, among others. The graph structure captures important information about the pairwise relationships between the features of the data. Therefore, algorithms that exploit the graph provide effective solutions for machine learning tasks where such structured data are considered.

Given the success of deep networks for learning representations from raw data over the last decade, there has been a significant increase in the development of end-to-end graph representation learning algorithms, commonly referred to as Graph Neural Networks (GNNs). The majority of GNN architectures are message-passing GNNs that take into account the graph geometry through the adjacency matrix [1] or the Laplacian matrix [2] of the graph and learn geometry-aware representations by learning the weights of linear combinations of the features of a node and its neighbors. Given the limited expressivity power of message passing GNNs [3], in the last two years new architectures that take into account more information about the geometry of the graph, either by considering higher order node interactions [4] or by using a positional encoding for each node [5], have been proposed.

In this thesis, our primary focus is the infusion of extra geometrical information in end-to-end deep learning algorithms for graphs. In order to achieve this, we leverage the mathematical theory of Optimal Transport (OT), which enables the definition of geometrically meaningful distances. Specifically, with OT one can use the geometry of a space, as captured by a distance metric, to define Wasserstein distances between probability measures on that space. In our work, we describe graph data as discrete probability measures and propose modules that use Wasserstein distances to compare them in a geometrically meaningful way. We use the recently

proposed entropic regularization of the OT problem and implicit differentiation to enable the automatic differentiation of our modules and, therefore, their integration in end-to-end GNN architectures. Further, for each of our proposed modules we analyse their behaviour, build intuitions and demonstrate with numerical experiments how the geometry awareness they offer enhances the representation learning process.

Representation learning for graphs arises in two different settings. First, the setting where the data is described by one graph. Such settings arise, most commonly, when each data point corresponds to a node of the graph and the edges describe some pairwise relationship among datapoints. An example of a machine learning task that may be relevant in that case is the classification of the nodes. The underlying geometry is the one that captures the relationship between the nodes. Second, the setting where the data consist of multiple graphs. In this setting there are two different types of pairwise relationships that need to be properly described: i) the relationships among the nodes of each graph and ii) the relationships between the graphs. Examples of machine learning tasks that may be relevant in that case include graph classification and graph regression. In this thesis we propose solutions that take into account the underlying geometry in both of these settings. We provide below an outline of our propositions.

1.2 Thesis Outline

The thesis is organized as follows:

In Chapter 2 we provide an introduction to Optimal Transport with a bias towards the entropy regularized OT problem, which is particularly relevant in machine learning. We gather the numerical methods that have been proposed for entropy-regularized OT problems and use them as a point of reference within the next chapters of the thesis. Further, we provide an overview of recent works in the literature of machine learning for graphs that employ elements from OT theory.

In Chapter 3 we consider the graph representation learning setting where the data are described by one, given graph. In that case, the geometry to be considered is fixed and corresponds to the pairwise relationships between the nodes. Therefore, we chose to describe the graph data as probability measures with fixed positions and capture their underlying geometry using the diffusion distance [6]. The problem that we are interested in is to learn interpretable node embeddings that are stable to perturbations of the graph structure. We propose an end-to-end differentiable autoencoder composed of an encoder with a linear layer and a decoder that uses OT to exploit the geometry. We cast the representation learning problem to that of learning simultaneously i) a low-dimensional space and ii) coordinates for the nodes in that space. We term our proposed architecture node2coords. The representations learned for the low-dimensional space are conducive to interpretability, as

they capture the most relevant geometric information of the underlying graph, and to stability, as they can be used to register the embeddings of perturbed graphs. The node coordinates are interpretable, as their values reveal the proximity to the representations that span the low-dimensional space. In that context, we propose a novel OT layer, that is incorporated in the decoder of node2coords. This layer is directly amenable to automatic differentiation and performs an interpolation in the Wasserstein space (aka a Wasserstein barycenter), of the graph data while using the geometry captured by the diffusion distance. The input to node2coords are node connectivity descriptors, which capture the local structure of the nodes. The node connectivity descriptors are passed through the encoder to learn a small set of graph structural patterns, which define the low-dimensional space. In the decoder, the node connectivity descriptors are reconstructed as Wasserstein barycenters of the graph structural patterns. The optimal weights for the barycenter representation of a node's connectivity descriptor correspond to the coordinates of that node in the low-dimensional space. Through experimental results we demonstrate that the representations learned with node2coords are stable to perturbations of the graph structure and achieve competitive or superior results compared to state-of-the-art unsupervised methods in the task of node classification. The content of this chapter has been published in [7].

In Chapter 4 we consider the setting where the data consist of multiple graphs. This setting is tackled by graph representation learning algorithms that use a set of training graphs in order to learn a mapping that can generalize to unseen, test graphs. An example of such a setting is when the graph data represent molecules, where nodes are atoms and edges the chemical bonds between them, and the task we are interested in is to classify them as active or inactive against cancer cells. End-to-end GNN architectures are common choices for such tasks. In GNN architectures the geometry between the nodes of each graph is taken into account with graph-aware layers, such as the MP layers discussed in Section 1.1. The graphs can naturally be of varying size. For instance, each molecule may be composed of a different number of atoms. Therefore, the problem we focus on in this chapter is how we can optimally pool the graph representations of varying size to a representation of fixed size that can be multiplied by the weights of the classifier that is driving the end-to-end representation learning process. In order to do so, we use OT to capture the geometry between the representations of different graphs. Since the GNN learns a mapping that generalizes to unseen test graphs, it follows that the graph representations can assume values anywhere within the representation space. Therefore, we chose to describe them as probability measures with free positions in the considered space and propose to pool a graph representation by minimizing the Wasserstein distance between itself and its pooled version. Our pooling method minimizes this distance by performing a Wasserstein gradient flow with respect to the pooled graph representation. Therefore, we term our proposed pooling method FlowPool. The minimization of the Wasserstein distance leads to the optimal preservation of the statistics of the graph representation to its pooled version. Further, using recently proposed methods of implicit differentiation, FlowPool lends itself to automatic differentiation. Through numerical experiments we show that our proposed

pooling method demonstrates promising results in the task of graph classification, when compared to state-of-the-art pooling methods.

1.3 Summary of Contributions

The main contributions of this thesis are summarized as follows:

- We propose a Wasserstein barycentric layer that can take into account relevant geometric information of the graph and that can be incorporated in end-to-end deep learning architectures for graphs.
- We integrate the Wasserstein barycentric layer to an autoencoder architecture and learn interpretable node embeddings that are stable to perturbations of the graph structure.
- We propose a pooling method that optimally preserves the statistics of a graph representation by minimizing the Wasserstein distance between itself and its pooled counterpart.
- We propose an effective method to perform automatic differentiation of the proposed pooling method and integrate it in end-to-end deep learning architectures for graph classification.

2 Optimal Transport for Machine Learning

In this chapter we provide an introduction to Optimal Transport (OT) and discuss aspects that are necessary for the comprehension of the next chapters of this thesis. We first provide an illustrative description of the optimal transportation problem. We then proceed to the mathematical formulation of the OT problems. Further, we discuss the entropy regularization that has majorly contributed to the adoption of Optimal Transport by the machine learning community in recent years. We gather the most relevant numerical methods that have been proposed in the literature for solving entropy-regularized optimal transport problems [8] in order to use them as a point of reference in Chapters 3 and 4. Finally, we provide an overview of the works that have been proposed to solve machine learning tasks on graphs using Optimal Transport.

2.1 Illustrative Description of The Optimal Transport Problem

The Optimal Transport problem was first formulated by Monge [9] in 1781 in order to find the optimal way to transport the entirety of a pile of sand to a different position. We describe this problem in an illustrative way in order to provide the reader with an intuitive visualization for it.

In Fig. (2.1), we depict with red the pile of sand μ that we want to move from the positions X and with blue the pile of sand ν, which will be the optimal displacement of μ at the positions Y. The amount of sand (or mass) at the position x is equal to $\mu(x)$. If the sand $\mu(x)$ is moved to position $y = T(x)$, as determined by a map T, the total effort for its displacement is considered to be equal to the distance $D(x, y)$ between the positions x, y multiplied by the amount of mass moved, $\mu(x)$. Therefore, the effort made to move the sand $\mu(x)$ is equal to $D(x, T(x))\mu(x)$. The objective of the optimal transport problem is to find the map T that will minimize the

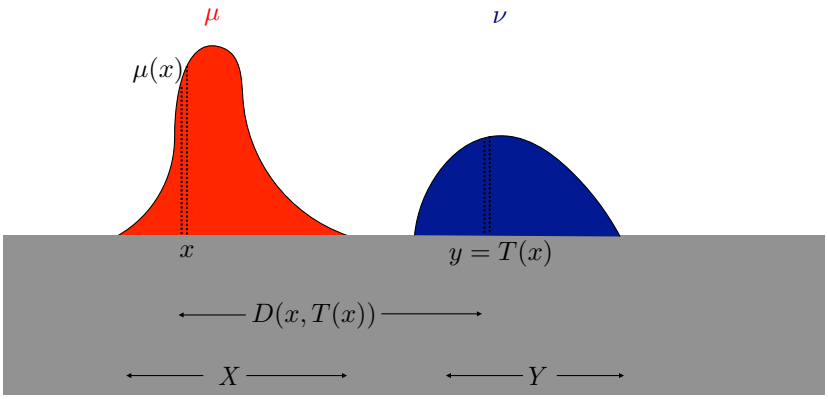


Figure 2.1 – The optimal transport problem between the source μ and target ν.

sum of these efforts for all amounts of sand $\mu(x)$ or equivalently:

$$\min_T \int_X D(x, T(x))\mu(x). \tag{2.1}$$

However, the reader will notice that there is still an important constraint that needs to be met according to Monge's problem formulation, namely to transport the *entirety* of the sand from pile μ to pile ν. In Fig. (2.2), we show that if the sand that is moved to B comes from the areas A_1 and A_2 of X, then it will hold that $\mu(A_1) + \mu(A_2) = \nu(B)$. Of course, this natural constraint must be satisfied for any area B of Y. For any B in Y, we can find the areas in X, from which the sand $\nu(B)$ was transferred, as $T^{-1}(B) = \{x | T(x) \in B\}$. As a result, the constraint is equivalent to:

$$\mu(T^{-1}(B)) = \nu(B), \forall B. \tag{2.2}$$

This constraint, that guarantees the preservation of the amount of sand from the pile μ to the pile ν, is called the pushforward constraint and is written as:

$$T_{\#}\mu = \nu. \tag{2.3}$$

Therefore, the Optimal Transport problem, as formulated by Monge, is the minimization in Eq.(2.1), subject to the constraint in Eq. (2.3). This problem found its place in probability

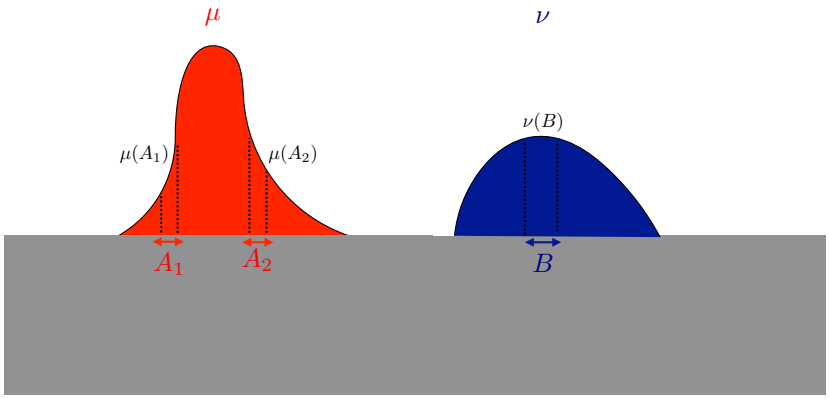


Figure 2.2 – The pushforward constraint.

theory by corresponding the piles of sand to probability measures. In that sense, the piles of sand μ, ν with equal mass, shown in Fig. (2.1), (2.2), can be thought of as continuous probability measures.

2.2 Optimal Transport Problems for Discrete Measures

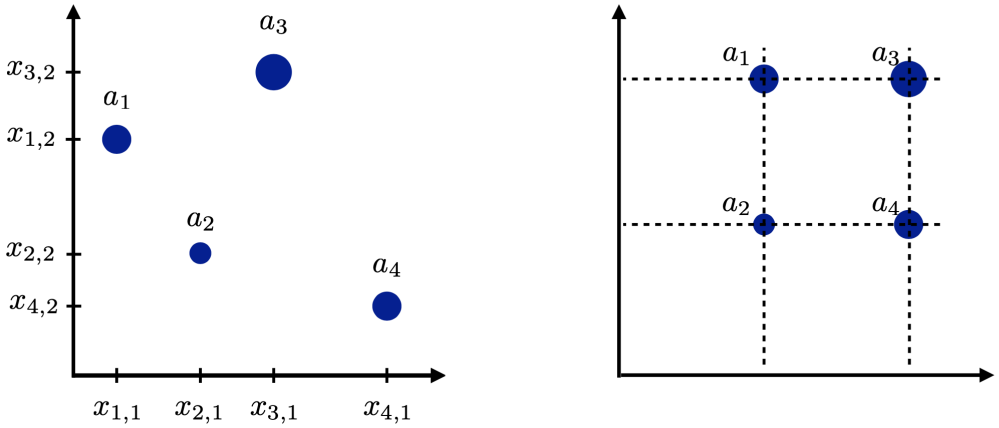
2.2.1 Discrete Measures

In this thesis we propose methods based on Optimal Transport that can be used for learning representations for graph-structured data. Since data are found in a discrete form, we focus on optimal transport problems for discrete measures. We consider the set of discrete measures $M(\Omega)$ defined on the space Ω. A measure $\mu \in M(\Omega)$ is of the form:

$$\mu = \sum_{i=1}^{n} a_i \delta_{x_i}, \tag{2.4}$$

where δ_{x_i} is the Dirac unit mass at the point x_i. Thus, the discrete measure μ is described by the vector of positive weights $a = [a_1, \ldots, a_n]$ and the set of points $X = (x_1, \ldots, x_n) \in \Omega$. In the case of discrete probability measures, the n-dimensional vector a is a histogram and belongs in the probability simplex $\Sigma_n = \{a \in \mathbb{R}_+^n | \sum_{i=1}^n a_i = 1\}$.

In Fig. (2.3) we show two examples of a discrete measure in a 2-dimesional space. The size of the i-th point is proportional to the weight a_i. In Fig. (2.3a) mass can be found anywhere in the 2-dimensional space. The position of the i-th point is defined by its coordinates $x_i = [x_{i,1}, x_{i,2}]$.



(a) Lagrangian discretization.

(b) Eulerian discretization.

Figure 2.3 – Examples of a discrete measure of four points. (a) Mass can be found anywhere in the space. (b) Mass can only be found at the positions predetermined by the fixed grid.

In Fig. (2.3b) mass can be found only at the positions determined by the fixed grid. The setting where mass can be found anywhere in the space Ω is referred to as a Lagrangian discretization. The setting where mass can be found only on a predefined set of points in Ω corresponds to an Eulerian discretization.

2.2.2 The Monge Problem

We consider a source measure $\mu = \sum_{i=1}^{n} a_i \delta_{x_i}$, a target measure $\nu = \sum_{j=1}^{m} b_j \delta_{y_j}$ and a cost matrix $C \in \mathbb{R}^{n \times m}$. The cost $C(x_i, y_j)$ quantifies the penalization for the transportation of mass from any point x_i in the support of measure μ to any point y_j in the support of the measure ν. The Monge problem aims to find a map T that minimizes the total cost of transportation:

$$\min_{T} \sum_{i=1}^{n} C(x_i, T(x_i)) \tag{2.5}$$

while satisfying the push-forward constraint:

$$b_j = \sum_{i:T(x_i)=y_j} a_i. \tag{2.6}$$

It can be directly seen that in the case where the number of points n of the source measure μ is smaller than the number of points m of the target measure ν, there exists no Monge map T that can transport μ to ν. This issue is alleviated with the relaxation of the transportation

problem, proposed by Kantorovich, which allows to split the mass.

2.2.3 The Kantorovich Problem

The relaxed transportation problem proposed by Kantorovich [10] in 1942 aims to find the optimal probabilistic coupling $P \in \mathbb{R}_+^{n\times m}$ between the measures μ and ν, instead of an optimal deterministic map T from μ to ν. In this setting, the element $P(i,j)$ of the coupling describes the probability of moving mass from the point x_i in the support of the measure μ to the point y_j in the support of the measure ν. The Kantorovich transportation problem between the measures $\mu = \sum_{i=1}^{n} a_i \delta_{x_i}$, $\nu = \sum_{j=1}^{m} b_j \delta_{y_j}$, and for cost matrix $C \in \mathbb{R}^{n\times m}$, searches for the optimal coupling P that minimizes the total cost of transportation:

$$\min_P \langle P, C\rangle \tag{2.7}$$

while satisfying the constraint:

$$P \in U(a,b) = \{P \in \mathbb{R}_+^{n\times m} | P\mathbb{1}_m = a \text{ and } P^\top \mathbb{1}_n = b\}. \tag{2.8}$$

The constraint in Eq. (2.8) guarantees that the entirety of the mass of measure μ, namely a, is transported to the mass of measure ν, namely b.

The problem defined in Eq. (2.7), (2.8) is a linear program. Its feasible region is the convex polytope $U(a,b)$ defined by the mass preservation constraints. It can be solved using linear programming solvers such as the simplex algorithm [11].

From now on we will denote the Kantorovich optimal transport problem between the measures $\mu = \sum_{i=1}^{n} a_i \delta_{x_i}$, $\nu = \sum_{j=1}^{m} b_j \delta_{y_j}$ as $L_C(a,b)$. This is done in order to highlight the dependency of the problem both on the geometry and the distribution of mass. The geometry is dictated by the supports of the measures $\{x_i\}_{i=1}^{n}$, $\{y_j\}_{j=1}^{m}$ and the cost C selected to penalize the transportation of mass from any x_i to any y_j. The distribution of mass is dictated by the weight vectors a and b. Therefore, from now on:

$$L_C(a,b) = \min_{P\in U(a,b)} \langle P, C\rangle. \tag{2.9}$$

2.2.4 Wasserstein Distance

The choice of the cost C influences in an important way the optimal transport problem. In the specific case where the cost C is the p-th power of a metric D on the space Ω, it is shown

through the Gluing Lemma (Theorem 7.3 in [12]) that $L_C(a,b)$ can be used to define a distance between the measures μ, ν. For instance, this is the case if the cost considered is the Euclidean distance, as in the illustrative example of Section 2.1.

If the cost $C = D^p$ for $p \geq 1$ satisfies the properties of a distance, namely, non-negativity, symmetry, the identity of indiscernibles and the triangle inequalities, then:

$$W_p(\mu,\nu) = L_{D^p}(a,b)^{\frac{1}{p}} \tag{2.10}$$

defines a distance between the measures μ,ν, called the p-Wasserstein distance. Therefore, optimal transport offers a principled way to use the distance between the support of the measures to define a distance between the measures themselves.

In the more general case, where the cost C is not a metric on Ω, $L_C(a,b)$ is not a distance. However, it can still be used to measure a similarity, or closeness, of the measures μ and ν, while taking into account the geometry of the space on which they are defined.

2.3 Entropy Regularization of the Optimal Transport Problem

As mentioned before, the Kantorovich problem defined in Eq. (2.9) is a linear program. As, a result, the complexity of computing $L_C(a,b)$ scales in at least $\mathcal{O}((n+m)nm\log(n+m))$ when comparing discrete measures of n and m points. Also, the value $L_C(a,b)$ is very susceptible to small changes in the measures μ and ν. When the values of the weights a,b undergo a small perturbation, the polytope $U(a,b)$ of the feasible couplings P will change. Since a linear program attains its minimum at a vertex of the feasible set, the optimal solution of the perturbed optimal transport problem can vary significantly from that of the original, unperturbed problem. Similarly, when the support $\{x_i\}_{i=1}^n$, $\{y_j\}_{j=1}^m$ of the measures μ,ν undergoes a small variation, the values of the cost matrix C change. This change results to the selection of different vertices of $U(a,b)$, and can therefore also affect significantly the solution of $L_C(a,b)$.

2.3.1 Entropy-Regularized Optimal Transport for Machine Learning

The issues of high computational complexity and instability to small perturbations hinder the application of optimal transport to machine learning and data science. For instance, discrete measures considered in machine learning applications can be composed of many datapoints, thus making the values of n and m large. Further, since real data may hold some uncerainty or noise, it is desirable to build machine learning models that are robust to this noise and do not overfit the data. Both of these issues are addressed by the entropic regularization of the optimal transport problem [13] proposed in the context of machine learning by Cuturi in 2013.

The entropy regularized optimal transport problem aims to find an optimal probabilistic coupling P between the measures μ and ν that minimizes the total cost of transportation and whose negative entropy is as small as specified by a regularization parameter ϵ. Given the entropy $H(P) = -\sum_{i=1}^{n}\sum_{j=1}^{m} P_{i,j}\log P_{i,j}$ of the coupling P, the regularized problem takes the form:

$$L_C^{\epsilon}(a,b) = \min_{P\in U(a,b)} \langle C, P\rangle - \epsilon H(P). \tag{2.11}$$

When the entropy regularization ϵ takes large values, the minimization of the negative entropy of the coupling dominates in Eq. (2.11) and therefore the optimal coupling is the independence matrix of the measures, which is equal to $ab^{\top}$. On the contrary, when ϵ takes small values, the solution of the entropy regularized problem $L_C^{\epsilon}(a,b)$ tends to that of the unregularized problem $L_C(a,b)$. Further, because the term $-H(P)$ is strictly convex with respect to P, the objective function in Eq. (2.11) is also strictly convex with respect to P and, therefore, there exists a unique optimal solution P^*. This is advantageous compared to the linear program in Eq. (2.9) which is convex and can have more than one optimal solutions.

We now discuss how entropic regularization alleviates the issues of high computational complexity and instability.

Reduction of Computational Complexity

Let $f \in \mathbb{R}^n$ and $g \in \mathbb{R}^m$ be dual variables. The Lagrangian $\mathscr{L}(P,f,g)$ of Eq. (2.11) is:

$$\mathscr{L}(P,f,g) = \sum_{i=1}^{n}\sum_{j=1}^{m}(P_{i,j}C_{i,j} + \epsilon P_{i,j}\log P_{i,j}) + f^{\top}(P\mathbb{1}_m - a) + g^{\top}(P^{\top}\mathbb{1}_n - b). \tag{2.12}$$

By taking the first order conditions we can obtain the following factorization for the probabilistic coupling P:

$$\frac{\partial\mathscr{L}(P,f,g)}{\partial P_{i,j}} = 0 \Rightarrow P_{i,j} = e^{\frac{f_i}{\epsilon}} e^{-\frac{C_{i,j}}{\epsilon}} e^{\frac{g_j}{\epsilon}}. \tag{2.13}$$

Therefore, by considering the non-negative vectors:

$$u = e^{\frac{f}{\epsilon}} \text{ and } v = e^{\frac{g}{\epsilon}}, \tag{2.14}$$

the optimal coupling P of Eq. (2.11) can be written as:

$$P = \text{diag}(u) K \, \text{diag}(v), \tag{2.15}$$

where :

$$K = e^{-\frac{C}{\epsilon}} \tag{2.16}$$

is the Gibbs kernel that corresponds to the cost matrix C and $\text{diag}(u), \text{diag}(v)$ denote the diagonal matrices obtained from the vectors u, v.

By substituting P, as factorized in Eq. (2.15), in the mass preservation constraints in Eq. (2.8) we obtain:

$$u \odot (Kv) = a \text{ and } v \odot (K^\top u) = b \tag{2.17}$$

where $\odot$ is the Hadamard (element-wise) product. By observing Eq. (2.17), one can see that the problem in Eq. (2.11) can be solved with a fixed point algorithm, by alternating the updates:

$$u^{(l+1)} = \frac{a}{Kv^{(l)}} \text{ and } v^{(l+1)} = \frac{b}{K^\top u^{(l+1)}} \tag{2.18}$$

where the divisions are applied element-wise and the initialization is a positive vector, for instance $v^{(0)} = \mathbb{1}_m$. The iterations in Eq. (2.18) define Sinkhorn's matrix scaling algorithm [14] on the non-negative matrix $K = e^{-\frac{C}{\epsilon}}$.

Therefore, the solution of the entropy-regularized problem with Sinkhorn's algorithm consists of matrix-vector multiplications and element-wise divisions and its computational complexity is $\mathcal{O}\left((n+m)^2\right)$ when comparing measures of n and m points. Finally, Sinkhorn's algorithm converges linearly and the convergence rate decreases as the entropy regularization parameter ϵ becomes smaller.

By substituting the values of the coupling P as a function of the dual potentials f, g as expressed in Eq. (2.13) into the Lagrangian of Eq. (2.12), we can obtain the dual formulation of the entropy-regularized transport problem as:

$$L_C^\epsilon(a,b) = \max_{f \in \mathbb{R}^n, g \in \mathbb{R}^m} \langle f, a \rangle + \langle g, b \rangle - \epsilon \langle e^{\frac{f}{\epsilon}}, K e^{\frac{g}{\epsilon}} \rangle. \tag{2.19}$$

The unconstrained maximization problem in Eq. (2.19) can be solved with block coordinate ascent [15]. The potentials f and g are updated alternatively by cancelling the respective gradients in these variables of the objective in Eq. (2.19).

Stability and Differentiability

We now focus on the stability of $L_C^\epsilon(a,b)$. What is meant by stability is that small perturbations of the input measures, either in their weights or their positions of mass, result in small changes of the optimal solution P^*. Equivalently, given an input measure $\mu = \sum_{i=1}^n a_i \delta_{x_i}$, it is desired that $L_C^\epsilon(a,b)$ is smooth with respect to the weights a and the positions of mass $\{x_i\}_{i=1}^n$.

Differentiation with respect to the weights

$L_C^\epsilon(a,b)$ is smooth and convex with respect to the weight vectors a, b. From the dual formulation in Eq. (2.19) it can be seen that the gradient of $L_C^\epsilon(a,b)$ with respect to a, b is:

$$\nabla L_C^\epsilon(a,b) = [f^*, g^*], \tag{2.20}$$

where $f^* = \epsilon \log(u) - \epsilon \frac{\log(u)^\top 1_n}{n} 1_n$ and $g^* = \epsilon \log(v) - \epsilon \frac{\log(v)^\top 1_m}{m} 1_m$ are the optimal potentials obtained from Eq. (2.14) and recentered so that they sum up to zero.

Differentiation with respect to the positions of mass

We remark that for $\epsilon > 0$, $L_C^\epsilon(a,b)$ is a smooth function of the cost matrix C with gradient:

$$\nabla_C L_C^\epsilon(a,b) = P^*, \tag{2.21}$$

where P^* is the unique optimal solution of Eq. (2.11) for the current value of the cost C or, equivalently, for the current positions of mass.

Furthermore, assuming that the space where the probability measures are defined is the d-dimensional Euclidean space $\Omega = \mathbb{R}^d$, so that $X = [x_1; \ldots, x_n] \in \mathbb{R}^{n \times d}$, $Y = [y_1; \ldots, y_m] \in \mathbb{R}^{m \times d}$, the gradient of $L_C^\epsilon(a,b)$ with respect to X can be obtained via the chain-rule as:

$$\nabla_X L_C^\epsilon(a,b) = \Big(\sum_{j=1}^m P_{i,j}^* \nabla_1 C(x_i, y_j) \Big)_{i=1}^n, \tag{2.22}$$

where ∇_1 denotes the gradient with respect to the first variable. Similarly, the gradient $\nabla_Y L_C^\epsilon(a,b)$ can be obtained using Eq. (2.21) and the gradient of the cost with respect to the second variable $\nabla_2 C(x_i, y_j)$.

2.3.2 Log-stabilized Solution

If the entropy regularization parameter ϵ is small compared to the entries of the cost matrix C, the values of the Gibbs kernel $K = e^{-\frac{C}{\epsilon}}$ become very close to zero. Therefore, the Sinkhorn iterations in Eq. (2.18) can lead to numerical overflow. This numerical issue can be alleviated by computing the Sinkhorn iterations in the log domain.

By considering the dual of the entropy-regularized optimal transport problem, the log-domain iterations can be obtained as [16]:

$$\begin{aligned} f^{(l+1)} &= \epsilon \log(a) - \epsilon \log\left(K e^{\frac{g^{(l)}}{\epsilon}}\right) \\ g^{(l+1)} &= \epsilon \log(b) - \epsilon \log\left(K^\top e^{\frac{f^{(l+1)}}{\epsilon}}\right). \end{aligned} \tag{2.23}$$

It can be noticed that, because of the relation between the dual potentials f, g and the scaling vectors u, v in Eq. (2.14), the iterations in Eq. (2.18) are equivalent to the iterations in Eq. (2.23). Therefore, in the case where there are no numerical overflows, the solution P^* obtained with the log domain computations is the same as that obtained with the unstabilized Sinkhorn iterations, up to numerical precision.

The optimal coupling P^* can be obtained from the optimal potentials f^*, g^* reached upon convergence as:

$$P^* = e^{\frac{f^* \mathbb{1}_m^\top + \mathbb{1}_n g^{*\top} - C}{\epsilon}}. \tag{2.24}$$

2.3.3 Relaxation of the Mass-Preservation Constraints

In some cases it may be desired to relax the mass preservation constraints in order to solve optimal transport problems for a source μ and a target ν that do not have the same mass. In [17], it is proposed to control the mass variation through a regularization parameter ρ, leading to the definition of the unbalanced optimal transport problem with entropy regularization:

$$L_C^\epsilon(a,b) = \min_{P \in \mathbb{R}_+^{n \times m}} \langle C, P \rangle - \epsilon H(P) + \rho\Big(\mathrm{KL}(P\mathbb{1}_m | a) + \mathrm{KL}(P^\top \mathbb{1}_n | b)\Big), \tag{2.25}$$

where $\mathrm{KL}(\cdot|\cdot)$ is the Kullback-Leibler divergence [18]. This problem can also be solved with Sinkhorn iterations as proposed in [17] with the scaling vectors u and v in Eq. (2.18) raised to the power $\frac{\rho}{\rho+\epsilon}$.

2.3.4 Metric Properties of the Entropy Regularized Transportation Problem

In the case where the cost matrix C in Eq. (2.11) is a metric, it can be shown through the Gluing Lemma that $L_C^\epsilon(a,b)$ satisfies the properties of non-negativity, symmetry and the triangle inequality (Theorem 1, [13]). Therefore, in the case where C belongs to the cone of distance matrices, $L_C^\epsilon(a,b)$ is called the Sinkhorn distance or the entropy regularized Wasserstein distance between the measures μ and ν. The term "distance" is used a bit loosely, in the sense that $L_C^\epsilon(a,b)$ does not satisfy the identity of indiscernibles.

From now on we will denote the entropy-regularized p-Wasserstein distance between measures $\mu = \sum_{i=1}^n a_i \delta_{x_i}$ and $\nu = \sum_{j=1}^m b_j \delta_{y_j}$ as $W_p^\epsilon(\mu,\nu)$. Further, in the specific case where measures are defined on fixed positions, as in the example of Fig. (2.3b) where the positions are defined by a grid, we will use simply $W_p^\epsilon(a,b)$.

2.4 Wasserstein Barycenter

Given S measures $\{\nu_i\}_{i=1}^S \in M(\Omega)$ their Wasserstein barycenter [19] is defined as the solution to the problem:

$$\underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{S} \lambda_i W_p^p(\mu,\nu_i) \tag{2.26}$$

subject to:

$$\sum_{i=1}^{S} \lambda_i = 1, \tag{2.27}$$

where $W_p(\mu,\nu_i)$ is the p-Wasserstein distance defined in Eq.(2.10). The barycenter is an interpolation in the Wasserstein space of the S measures $\{\nu_i\}_{i=1}^S$ with weights $\{\lambda_i\}_{i=1}^S$. The weights $\{\lambda_i\}_{i=1}^S$ are commonly referred to as barycentric weights or barycentric coordinates. Barycenters can also be defined for the entropy-regularized Wasserstein distance, discussed in Section 2.3, by substituting in Eq. (2.26) W_p by W_p^ϵ.

2.4.1 Wasserstein Barycenter of Fixed Support

In the specific case where all the considered measures are defined on n fixed positions $\{x_j\}_{j=1}^n$, the Wasserstein interpolation amounts to the computation of the weights a of the unknown barycenter measure $\mu = \sum_{j=1}^n a_j \delta_{x_j}$. Given, the histograms b_i of the measures ν_i, the

Wasserstein barycenter problem amounts to:

$$\hat{a} = \underset{a \in \Sigma_n}{\operatorname{argmin}} \sum_{i=1}^{S} \lambda_i W_p^p(a, b_i) \tag{2.28}$$

subject to:

$$\sum_{i=1}^{S} \lambda_i = 1. \tag{2.29}$$

2.4.2 Entropy-Regularized Wasserstein Barycenter of Fixed Support

Entropy-regularized Wasserstein barycenters for measures with fixed positions of mass can be computed using Sinkhorn iterations. When computing the barycenter of S histograms we are solving simultaneously S optimal transport problems between each of the S known targets, which are the S histograms $\{b_i\}_{i=1}^{S}$, and the unknown source, which is the barycenter a. Therefore, for the entropy-regularized case described above, S sets of scaling vectors u and v have to be computed. The computation of the entropy-regularized, Wasserstein barycenter can be performed through Sinkhorn iterations [20]. An extra step is added for the estimation of the unknown barycenter a, which is needed for the update of the second scaling vectors [21]. The solution of the entropy-regularized Wasserstein distance with Sinkhorn iterations is equivalent to the Kullback-Leibler (KL) [18] projection of the transportation coupling P to the convex sets defined by the mass preservation constraints [21]. The expression for the estimation of the unknown barycenter is derived from the first order conditions of the KL projections of the couplings to the convex sets defined by the mass preservation constraints of the known targets $\{b_i\}_{i=1}^{S}$ [21].

In the specific case where the Wasserstein distance uses the mass relaxation of Eq. (2.25) the obtained barycenter is the unbalanced Wasserstein barycenter. Unbalanced Wasserstein barycenters [17] have been shown to better preserve the shape of the histograms $\{b_i\}_{i=1}^{S}$ because erroneous mass does not have to appear in the barycenter in order to satisfy the mass preservation constraints. The L Sinkhorn iterations for the computation of the unbalanced Wasserstein barycenter are shown in Algorithm 1.

2.5 Optimal Transport for Machine Learning on Graphs

Optimal transport is increasingly being used in machine learning. In this section we review methods that are based on optimal transport and that have been proposed in order to tackle a variety of machine learning tasks on graphs.

Algorithm 1 Unbalanced Wasserstein Barycenter of Fixed Support

Input: $\{b_i\}_{i=1}^{S}, \{\lambda_i\}_{i=1}^{S}$
Initialization
for $i \leftarrow 1$ to S **do**
  $u_i^{(0)} = 1_n$
end for
for $l \leftarrow 0$ to $L-1$ **do**
  Update first scaling vectors
  for $i \leftarrow 1$ to S **do**
    $$v_i^{(l)} = \left(\frac{b_i}{K^\top u_i^{(l)}} \right)^{\frac{\rho}{\rho+\epsilon}}$$
  end for
  Estimate Barycenter
  $$a^{(l)} = \left(\sum_{i=1}^{S} \lambda_i (u_i^{(l)} \odot K v_i^{(l)})^{\frac{\epsilon}{\epsilon+\rho}} \right)^{\frac{\epsilon+\rho}{\epsilon}}$$
  Update second scaling vectors
  for $i \leftarrow 1$ to S **do**
    $$u_i^{(l+1)} = \left(\frac{a^{(l)}}{K v_i^{(l)}} \right)^{\frac{\rho}{\rho+\epsilon}}$$
  end for
end for
return $\hat{a} = a^{(L-1)}$

DVNE [22] is an unsupervised learning algorithm for graphs that proposes a Wasserstein-based loss for training. The authors introduce an autoencoder that learns Gaussian distributions in the Wasserstein space as the latent representations of the nodes. They employ a Wasserstein-based loss function in order to guarantee the preservation of the first-order and the second-order node proximities in the latent representation space. In [23] the authors define a graph kernel between a pair of two graphs. They use the Weisfeiler-Lehman graph kernel [24], [25] to obtain the graph embeddings and then compute their Wasserstein distance, using as a cost the Hamming distance for categorical node features and the squared Euclidean distance for continuous features. Their proposed Wasserstein Weisfeiler-Lehman graph kernel is then defined using the computed pairwise Wasserstein distances.

A line of works employ the Gromov-Wasserstein distance in order to compute meaningful distances between graphs. The Gromov-Wasserstein distance [26] can be used to compare metric spaces that are equipped with a probability distribution. In [27] this distance is extended to define a distance between similarity matrices. The following works use Gromov-Wasserstein distances between the adjacency matrices of graphs.

In [28] the authors propose a fusion of Wasserstein and Gromov–Wasserstein distances in order to encode simultaneously both feature and structural information. The Wasserstein distance is used to measure similarities of the feature representations and the

Gromov-Wasserstein distance is used to measure the similarities of the graph topologies. The proposed fused distance is used to perform a variety of tasks, such as graph classification, graph clustering and the interpolation of graph structures. In [29] a scheme based on Gromov-Wasserstein that learns simultaneously the node embeddings and the optimal matching between two graphs is proposed. The scheme boils down to an optimization problem that is solved with an iterative process, where i) the node embeddings are used to estimate distance matrices for the matching and ii) the learned matching regularizes the node embeddings in the next iteration. The proposed scheme is validated on graph alignment and recommendation tasks. In [30] the authors propose to use a Gromov-Wasserstein barycenter of a set of graphs to perform graph partitioning and graph matching. The barycenter graph is a disconnected graph and the optimal transport from the barycenter graph to any other graph indicates a clustering structure. They propose an efficient regularized proximal gradient algorithm. In [31] a dictionary learning algorithm for graphs of varying size is introduced. The authors propose a factorization where the adjacency matrices of the graphs are expressed as Gromov-Wasserstein barycenters of the atoms in the dictionary. They propose two different implementations of the Gromov-Wasserstein factorization module, one with the proximal point algorithm (PPA) and one with Bregman alternating direction method of multipliers (BADMM) [32] and report results on graph clustering. An online graph dictionary learning algorithm is proposed in [33]. The authors introduce an efficient stochastic algorithm to learn a dictionary for graphs of different sizes and use the Gromov-Wasserstein distance as a data fitting term. However, in contrast to [32], they use a linear representation of the atoms instead of a barycenter. Therefore, their algorithm enjoys a smaller computational complexity. They report results for online graph subspace estimation and tracking. In [34] a framework to perform cross-domain alignment is proposed. Each domain is represented as a graph and a neural network is trained that minimizes the Gromov-Wasserstein distance between the graph structures and the Wasserstein distance between the features. The proposed framework is validated experimentally on tasks like image and text retrieval, visual question answering, image captioning, machine translation, and text summarization.

A different approach to compare graphs is proposed in [35], where the problem of defining a distance between graphs is cast to that of defining a distance between the distribution of smooth signals defined on graphs. The authors consider graphs of the same size and optimize for a permutation matrix that aligns the graphs. They report results on graph alignment and graph classification. The work in [36] builds on the work in [35]. The authors optimize for a probabilistic coupling from the vertices of one graph to the other in order to allow for the comparison of graphs of different sizes. Further, they show that the defined distance is a metric on the set of isomorphism classes of finite graphs and report results on graph sketching, graph retrieval and graph summarization.

More recently, in [37] the authors propose to learn hierarchical abstractions of graph structures in an unsupervised manner. They use a coarsening approach of the graph, based on Algebraic

Multigrid [38], and parametrize the coarsening matrices using a graph convolutional layer. The sum of the Wasserstein distances of the graphs from their coarsened versions is used as the loss function to train the network. Further, in [39] the authors propose two effective log-linear time approximations of the transportation cost matrix. The first is a sparse approximation based on locality sensitive hashing and the second a Nyström approximation with LSH-based sparse corrections. The proposed approximations are employed in order to perform in an efficient way graph distance regression with a siamese GNN-based architecture.

In our work we propose differentiable modules that solve OT problems and can be incorporated in end-to-end deep learning architectures for graph representation learning. In Chapter 3 we propose a module that computes barycenters of histograms defined on a graph and we incorporate it in an autoencoder in order to learn node embeddings that are directly interpretable and stable to perturbations of the graph structure. In Chapter 4 we propose a differentiable module that performs a Wasserstein gradient flow for graph representations of varying size. We incorporate this module in GNN architectures in order to perform global pooling of graphs, while optimally preserving the statistics of the graph representations.

3 Graph Representation Learning with Wasserstein Barycenters

3.1 Introduction

In this chapter [1], we focus on the case where we are interested in learning representations for a given graph. This case is relevant in machine learning settings where the datapoints under consideration are related in some way. As an example, in a citation network, each node corresponds to a publication and an edge exists between two nodes if either publication has cited the other one. Therefore, in order to perform machine learning tasks, such as node classification, it is necessary to design algorithms that learn effective representations for the nodes.

Existing graph representation learning methods have, in principle, two limitations. First, they are relatively unstable to perturbations of the graph structure [40]. Second, even if their design is justified intuitively, they often do not lead to directly interpretable node embeddings [41], [42]. In particular, there is not a clear interpretation of the values in a node embedding vector. In this work, we address these two limitations by proposing node2coords, an unsupervised learning algorithm that, given a graph structure, learns simultaneously a low-dimensional space as well as node coordinates in that space. The low-dimensional space is conducive to stability because it provides a way of registering the representations of perturbed graphs. The values of the node coordinates are interpretable and they correspond to the proximity to the representations that define the low-dimensional space.

We propose an autoencoder architecture with a Wasserstein barycentric decoder. The input to the algorithm are node connectivity descriptors, which capture the local structure of the nodes. In the encoding step, the node connectivity descriptors pass through a linear layer to obtain a set of graph patterns. This set of patterns defines the low dimensional space and provides essentially a compressed version of the structural information in the graph. In the decoding

[1] This work was published in a slightly different form in:
E. Simou, D. Thanou, and P. Frossard, "node2coords: Graph representation learning with wasserstein barycenters," IEEE Transactions on Signal and Information Processing over Networks, vol. 7, pp. 17–29, 2020.

step, each node connectivity descriptor is optimally reconstructed as a Wasserstein barycenter of the graph structural patterns by learning its barycentric coordinates [43]. The barycentric coordinates of a node are used as its representation. Therefore, the value of each feature in the representation of a node can be interpreted as the proximity, in terms of Wasserstein distance, of the node connectivity descriptor to the corresponding graph pattern.

We perform experiments on synthetic and real data and demonstrate that the node embeddings obtained with node2coords are stable with respect to perturbations of the graph structure. Furthermore, we show that in the task of node classification we achieve competitive or superior results compared to state-of-the art unsupervised learning algorithms that leverage only the graph structure [44].

The organization of this chapter is as follows. In Section 3.2 we review the related work in unsupervised learning of graph representations. In Section 3.3 we present in detail our Wasserstein barycenter representation method, which is incorporated in the decoder of node2coords. We outline our proposed autoencoder architecture in Section 3.4 and provide explanations for the representations it learns. In Section 3.5 we evaluate through experiments the performance of node2coords in machine learning tasks and we conclude in Section 3.6.

3.2 Related Work

The design of algorithms that learn representations for graph structured data has been extensively researched in the last years. Such algorithms learn low-dimensional representations for the nodes of the graph, which capture the most important information for inference on the graph. Given the large body of works on learning graph representations, we focus here on the ones that, similary to node2coords, learn graph representations in an unsupervised manner and that have been designed to use only the graph structure as the input to the algorithm. Such algorithms can be grouped into four categories, namely distance-based methods, matrix factorization methods, skip- gram methods and autoencoders.

Distance-based embedding methods learn an embedding look-up by forcing nodes that are close in the graph to be mapped as close as possible in the embedding space, with respect to a chosen distance metric. Notable methods in this category are Isomap [45], Locally Linear Embedding [46] and Laplacian Eigenmaps [47]. These methods capture the proximity of the nodes on the graph via the geodesic distance, a linear approximation of the local neighborhood of the nodes and the smoothness of the eigenvectors of the Laplacian matrix of the graph, respectively.

Matrix factorization methods aim to learn low-rank representations of the adjacency matrix of a graph. The first such work is Graph Factorization [48] where an efficient distributed

algorithm is proposed for the factorization of large graphs. This is followed by HOPE [49] which proposes matrix factorization for directed graphs and GraRep [50] which allows to capture higher order node proximities by considering powers of the adjacency matrix.

Skip-gram graph embedding methods are inspired by word embedding methods that use skip-gram [51] in order to predict context words for a given target word. Thus, by creating sequences of nodes, similarly to sentences of words, it is possible to learn embeddings by maximizing the probability of context nodes. Sequences of nodes are generated using random walks and the obtained node sequences are fed to a skip-gram model which maximizes their log-likelihood and provides the node embeddings. In Deepwalk [52] a neural network is trained by maximizing the probability of predicting context nodes for each node of the graph. Node2vec [53] creates node sequences by generating biased random walks. It combines Breadth First Search (BFS) and Depth First Search (DFS) [54] in order to learn embeddings that capture similarities of the nodes in terms of local structure as well as homophily. LINE [55] learns embeddings that preserve first-order and second-order node proximities.

In the autoencoder architectures proposed for graph representation learning, the encoder, most commonly, takes as input the adjacency matrix of the graph and outputs the node embeddings. The decoder uses these embeddings to reconstruct the adjacency matrix. SDNE [56] learns embeddings that preserve first-order and second-order node proximities. In order for this to be achieved, a regularizer forces nodes that are connected on the graph to be close in the embedding space. DNGR [57] creates a similarity matrix from the graph adjacency matrix using random surfing [58], a probabilistic method that employs random walks. The similarity matrix is fed to a denoising autoencoder in order to obtain the node embeddings. DVNE [22] maps nodes to Gaussian distributions in order to account for uncertainties in the graph structure and imposes neighboring nodes to have a small Wasserstein distance between their corresponding Gaussians. DVNE, in terms of architecture, is an autoencoder composed of linear layers at the encoder and the decoder and only employs a Wasserstein-based loss at the objective function. In contrast, in node2coords we propose an autoencoder with a non-linear Wasserstein barycentric layer at the decoder.

It is worth noting that Wasserstein barycenters have been used for representation learning in different contexts. For example, in [59] the authors propose a dictionary learning algorithm where images are reconstructed as Wasserstein barycenters of the atoms in the dictionary. Further, Gromov-Wasserstein barycenters have been employed in the context of graph representation learning in the works of [30], [31], discussed in Section 2.5.

3.3 Wasserstein Barycenters for Graph Representation Learning

In this section we propose a graph Wasserstein barycenter representation method, which provides differentiable, geometry-aware non-linearities and can be incorporated in different

deep network architectures. We first present our efficient implementation for the parallel computation of multiple Wasserstein barycenters on graphs and analyse its time complexity. Further, we provide an illustrative example of the non-linear, geometry aware interpolation obtained with Wasserstein barycenters and provide intuitions of why they were used in our graph representation learning algorithm.

3.3.1 Efficient Method for Barycenter Computation of Graph Patterns

Given a graph G of N nodes with adjacency matrix A, the input to our method can be any set of S non-negative, N-dimensional vectors with l_1 norm equal to one. From now on we will refer to such vectors, input to our proposed graph Wasserstein barycenter computation method, as *graph patterns*.

Our method takes as input a matrix of S graph patterns $M_S = [m_1, \ldots, m_S]$, with m_i an $N \times 1$ graph pattern, and outputs their J Wasserstein barycenters $B_J = [b_1, \ldots, b_J]$, as computed for J sets of barycentric coordinates Λ given by:

$$\Lambda = \begin{bmatrix} \lambda_{1,1} & \ldots & \lambda_{1,S} \\ \ldots\ldots\ldots\ldots & & \\ \lambda_{J,1} & \ldots & \lambda_{J,S} \end{bmatrix}. \tag{3.1}$$

The barycenter b_i is an N-dimensional vector obtained as the barycenter of the graph patterns in M_S with weights $\Lambda(i,\cdot) = [\lambda_{i,1}, \ldots, \lambda_{i,S}]$. Therefore, when learning graph representations with Wasserstein barycenters, the parameters being learnt are the barycentric coordinates in Λ.

We propose to compute Wasserstein barycenters for patterns defined on a graph by taking into account the underlying graph geometry through the diffusion distance D_τ [6]. Hence, the geometry aware cost C is chosen to be $C = D_\tau^{p=1}$ and the Gibbs kernel becomes $K = e^{-\frac{D_\tau}{\epsilon}}$. The diffusion distance D_τ captures the similarity of node connections in τ hops and is computed using the τ-th power of a Markov matrix P defining a random walk on the graph. For the graph G with adjacency matrix A, the degree of node i is defined as:

$$d(i) = \sum_{j=1}^{N} A(i,j) \tag{3.2}$$

and the Markov matrix P as:

$$P(i,j) = \frac{A(i,j)}{d(i)}. \tag{3.3}$$

The diffusion distance D_τ between a pair of nodes i, j is computed as:

$$\begin{aligned} D_\tau^2(i,j) &= \|P^\tau(i,\cdot) - P^\tau(j,\cdot)\|_{L^2}^2 \\ &= \sum_{u=1}^{N} \frac{(P^\tau(i,u) - P^\tau(j,u))^2}{\pi(u)} \end{aligned} \tag{3.4}$$

where:

$$\pi(x) = \frac{d(x)}{\sum_{y=1}^{N} d(y)}. \tag{3.5}$$

When two nodes i, j have similar connections in their τ-hop neighborhood, the diffusion distance $D_\tau(i,j)$ has a small value. The greater the difference of the τ-hop connectivities of two nodes, the larger the value of their diffusion distance.

As mentioned in Chapter 2, the choice of the cost C used for the transportation of mass is very important as it directly impacts the way that the underlying geometry is taken into account. As a result, the choice of C affects also the representations learned with Wasserstein barycenters. The diffusion distance cost D_τ enables a geometric analysis of the underlying graph structure at different scales through the choice of the parameter τ. We select the values of the parameter $\tau \in \mathbb{Z}_+$ in the set $[1, d_{max}]$, where d_{max} is the diameter of the graph. The diameter of the graph is equal to the greatest shortest path distance between any pair of vertices [60]. In the case where the graph under consideration exhibits a clustered structure, it is reasonable to select values for τ such that τ / d_{max} is small and τ is comparable to the maximal shortest path distance between nodes belonging to the same cluster. By doing so, the diffusion distance $D_\tau(i,j)$ for two nodes i, j that belong to the same cluster has a small value and, as a result, the matrix D_τ highlights the clusters of the graph. In the case of graphs that do not have clusters, values of τ that are closer to d_{max} are relevant, as they allow to capture the similarities of node connections at the scale of the entire graph.

Given the Gibbs kernel $K = e^{-\frac{C}{\epsilon}}$, we can compute unbalanced Wasserstein barycenters with Sinkhorn iterations as presented in Section 2.4. However, at each Sinkhorn iteration it is needed to update each of the S scaling vectors v, and then, once the barycenter has been estimated, update each of the S scaling vectors u. In the case where the updates of the S sets of scaling vectors are performed serially, the computation of the barycenter is inefficient in terms of time complexity, because of the nested loops.

We avoid this increase in the time complexity with S by proposing a parallelized and computationally efficient method for the barycenter computation. Specifically, we update in parallel the S scaling vectors v. Then, after the barycenter estimation, we also update in parallel the S scaling vectors u. Our proposal for the efficient Wasserstein barycenter computation is demonstrated in Algorithm 2. An important element of the parallelization is

due to the fact that the matrix-vector multiplications of the transpose of the Gibbs kernel $K^\top$ with the scaling vectors u_k can be implemented in parallel as a matrix-matrix multiplication of the $N \times N$ matrix $K^\top$ with the $N \times S$ matrix $U = [u_1, \ldots, u_S]$ whose columns are the S scaling vectors $\{u_k\}_{k=1}^S$ [13]. Similarly, the matrix-vector multiplications of the matrix K with the S scaling vectors $\{v_k\}_{k=1}^S$ can be implemented in parallel as a matrix-matrix multiplication of K with $V = [v_1, \ldots, v_S]$. We demonstrate in Algorithm 2 our proposed method for the computation of Wasserstein barycenters in parallel.

Algorithm 2 Barycenter Computation in node2coords

Input: $M_S, \Lambda(i,\cdot)$

Initialisation :

$U^{(0)} = 1_{N \times S}$

for $l = 0$ to $L-1$ **do**

  Update first scaling vectors

  $V^{(l)} = \left(\frac{M_S}{K^\top U^{(l)}}\right)^{\frac{\rho}{\rho+\epsilon}}$

  Estimate Barycenter

  $B_J(i,\cdot)^{(l)} = \left(\left((1_N \otimes \Lambda(i,\cdot)) \odot (U^{(l)} \odot KV^{(l)})^{\frac{\epsilon}{\epsilon+\rho}}\right) 1_S\right)^{\frac{\epsilon+\rho}{\epsilon}}$

  Update second scaling vectors

  $U^{(l+1)} = \left(\frac{B_J(i,\cdot)^{(l)} \otimes 1_S}{KV^{(l)}}\right)^{\frac{\rho}{\rho+\epsilon}}$

end for

return $B_J(i,\cdot)^{(L-1)}$

We now analyse step by step the time complexity of our proposed method. The first step is the update of the scaling vectors in V. In our implementation, the update of the S scaling vectors V is equivalent to the matrix multiplication $K^\top U$, the element-wise division of the $N \times S$ matrices M_S and $K^\top U$ and the elementwise exponentiation of the resulting $N \times S$ matrix to $\frac{\rho}{\rho+\epsilon}$. The complexity for the update of the scaling vectors V is therefore $\mathcal{O}(N^2S + 2NS)$.

The second step is the estimation of the Wasserstein barycenter. We perform this update efficiently using matrix operations as:

$$B_J(i,\cdot)^{(l)} = \left(\left((1_N \otimes \Lambda(i,\cdot)) \odot (U^{(l)} \odot KV^{(l)})^{\frac{\epsilon}{\epsilon+\rho}}\right) 1_S\right)^{\frac{\epsilon+\rho}{\epsilon}}. \tag{3.6}$$

The time complexity of this operation is $\mathcal{O}(N^2S + 5NS + N)$.

The third and final step is the update of the S scaling vectors U. This step is equivalent to the matrix multiplication KV, the element-wise division of the $N \times S$ matrices $B_J(i,\cdot) \otimes 1_S$ and KV and the elementwise exponentiation of the resulting $N \times S$ matrix to $\frac{\rho}{\rho+\epsilon}$. Thus, the complexity for the update of the scaling vectors U is $\mathcal{O}(N^2S + 3NS)$.

As a result, the time complexity for our implementation of a Sinkhorn iteration of the barycenter $B_J(i,\cdot)$ is $\mathcal{O}(3N^2S + 10NS + N)$, which means that it scales quadratically with respect to the number of nodes $\mathcal{O}(N^2)$. The J barycenters in B_J can be computed simultaneously using broadcasting operations, which are common in libraries such as PyTorch [61], and the complexity of each Sinkhorn iteration for the computation in parallel of J barycenters is $\mathcal{O}(JN^2S)$. Therefore, the overall complexity of the barycenter computation, which is composed of L Sinkhorn iterations, is $\mathcal{O}(LJN^2S)$. The number of Sinkhorn iterations L needed in order for the barycenter computation to converge increases as the entropy regularization parameter ϵ decreases [62].

3.3.2 Illustrative Example of Wasserstein Barycenter of Graph Patterns

In this section we provide an illustration of the representations obtained with Wasserstein barycenters of graph patterns. Consider a graph composed of two clusters. Let m_1 and m_2 be two graph patterns localized at each cluster of the graph. The patterns m_1, m_2 as well as their unbalanced Wasserstein barycenter b for $\lambda_1 = 0.2$ and $\lambda_2 = 0.8$ are shown in Fig. (3.1) [2]. We note that m_1 and m_2 are N-dimensional patterns defined on a graph. Their barycenter b is an interpolation that takes into account the graph through the diffusion distance cost $C = D_\tau$. In Fig. (3.1) we plot m_1, m_2 and b on the graph in order to highlight that the Wasserstein barycenter b is a geometry-aware non-linear interpolation of m_1 and m_2. It can be seen that the values of the barycentric coordinates λ_1, λ_2 quantify the proximity of the barycenter b with respect to the patterns m_1, m_2. Also, it can be seen that the barycenter b has a larger support than the patterns m_1 and m_2, that are being interpolated, because of the entropy regularization. Specifically, as the value of the entropy regularization parameter ϵ becomes larger, the barycenter tends to be uniform over the graph.

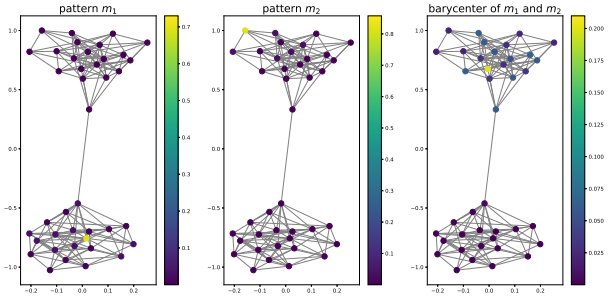


Figure 3.1 – Localized graph patterns m_1, m_2 plotted on the graph and their Wasserstein barycenter b for $\lambda_1 = 0.2$ and $\lambda_2 = 0.8$. m_1, m_2 and b are plotted on the graph in order to highlight that the barycentric interpolation takes into account the underlying graph.

In our proposed graph representation learning algorithm node2coords, graph patterns similar to those illustrated in Fig. (3.1) are interpolated at the Wasserstein barycentric decoder. We

[2] For the illustrations in Fig. (3.1), (3.3), (3.4), (3.5), (3.6), (3.10) we used the PyGSP toolbox [63].

note finally that the graph patterns that can be interpolated with Wasserstein barycenters are not restricted to be localized, as those shown in Fig. (3.1). Our proposed method can be integrated in different algorithms for learning representations of graph structured data by providing geometry aware, non-linear interpolations.

3.4 node2coords

Our proposed unsupervised graph representation learning algorithm node2coords relies on the graph Wasserstein barycenter representation method, introduced in Section 3.3. The proposed autoencoder architecture is shown in Fig. (3.2). The input of the encoder are the connectivity descriptors of the nodes Z_n, which capture their local structure. The node connectivity descriptors are passed through a linear layer followed by a softmax activation in order to obtain the small set of graph structural patterns M_S. In the decoder we employ Wasserstein barycenters as demonstrated in Section 3.3, to reconstruct the node connectivity descriptors as Wasserstein barycenters of the graph patterns in M_S. Thus, node2coords learns both the graph patterns M_S and the barycentric coordinates Λ. We give more details of each block of node2coords below.

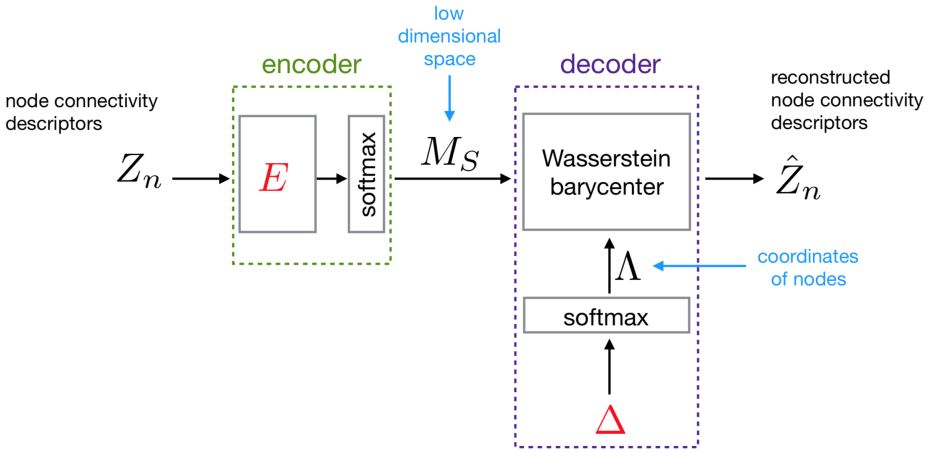


Figure 3.2 – Node2coords block scheme. In the encoder, the node connectivity descriptors are passed through a linear layer followed by a softmax activation to obtain the small set of graph structural patterns that define the low dimensional space M_S. In the decoder, the node connectivity descriptors are reconstructed as Wasserstein barycenters of the patterns in M_S by optimizing for their barycentric coordinates Λ. The barycentric coordinates are re-parametrized through a softmax layer in order to guarantee that they sum up to one for each node. The learned parameters are the weights of the encoder E and the weights of the decoder Δ, which are annotated with red.

Input: Given the adjacency matrix A of a graph of N nodes, we define the matrix of node

connectivity descriptors Z_n as:

$$Z_n(i,j) = \frac{\tilde{A}^n(i,j)}{\sum_{j=1}^{N} \tilde{A}^n(i,j)}. \tag{3.7}$$

The matrix $\tilde{A}$ is defined as $\tilde{A} = A + \alpha I_N$, where I_N is the N-dimensional identity matrix and $\alpha \in \{0,1\}$. The matrix $\tilde{A}^n$ is therefore computed as $\tilde{A}^n = (A + \alpha I_N)^n$. The i-th row $Z_n(i,\cdot)$ is the connectivity descriptor of node i and its support indicates the nodes that can be reached from node i in up to n hops. The value of the parameter α is set to $\alpha = 0$ for $n = 1$ because for $n = 1$ the only nodes that can be reached from a node are its one-hop neighbors. However, for $n \geq 2$ the value of α is set to $\alpha = 1$. The reason for this choice is that for $n \geq 2$, a node can always reach itself by hoping to its first-hop neighbors and back. The value of the parameter n depends on the size of the graph G. Specifically, for larger values of the number of nodes N, larger values for n are required. An example of a connectivity descriptor for $n = 1$ plotted on the graph is shown in Fig. (3.4).

Encoder: In the encoder, the node connectivity descriptors Z_n are passed through a linear layer with $N \times S$ ($S < N$) parameters E followed by a softmax activation. The $N \times S$ matrix M_S obtained at the output of the encoder is therefore:

$$M_S = \text{softmax}(Z_n E), \tag{3.8}$$

where the softmax activation of an N-dimensional vector x is obtained as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}. \tag{3.9}$$

The S patterns in M_S capture the most important structural properties of the graph and, therefore, we refer to them as graph structural patterns.

Decoder: In the decoder, the node connectivity descriptors in Z_n are reconstructed as Wasserstein barycenters of the S graph structural patterns in M_S using Wasserstein barycenters, as introduced in Algorithm 3.3. We obtain the optimal representations of the connectivity descriptors in $\hat{Z}_n$ as Wasserstein barycenters of M_S by learning their barycentric coordinates Λ.

In order to guarantee that the barycentric coordinates of each of the barycenter representations sum up to one, we introduce a change of variable, through a softmax, so that the barycentric coordinates $\Lambda(i,\cdot)$ of the barycenter approximation of the i-th node connectivity descriptor $Z_n(i,\cdot)$ are reparametrized through a matrix Δ as:

$$\lambda_{i,k} = \frac{e^{\delta_{i,k}}}{\sum_{j=1}^{S} e^{\delta_{i,j}}}. \tag{3.10}$$

The node connectivity descriptors in Z_n, that are being approximated as Wasserstein barycenters at the decoder, are localized in the n-hop neighborhood of the nodes. Therefore, their barycenter approximations in $\hat{Z}_n$ are also localized in the n-hop neighborhood of the nodes. The localization of the barycenters in $\hat{Z}_n$ in n hops, leads to learning patterns in M_S that are localized in up to n-hops. As the entropy regularization parameter ϵ decreases, the patterns in M_S tend to be localized in exactly n hops. On the contrary, as ϵ increases the patterns in M_S become more localized. We note, however, that it is not possible to control the nodes on which the graph structural patterns will have their highest values. Furthermore, in the Wasserstein barycentric layer of the decoder the graph is taken into account through the diffusion distance cost $C = D_\tau^{p=1}$. The nature of the displacement interpolation obtained with Wasserstein barycenters [64] and the use of the diffusion distance cost, which captures the geometry of the underlying graph, leads to a small set of patterns in M_S that highlight its structural properties.

It is important to observe that the decoding step can be thought of as an embedding of the nodes in the space spanned by the patterns in M_S. The i-th node of the graph, as described by its connectivity $Z_n(i,\cdot)$, is embedded in the S-dimensional space defined by the patterns in M_S by learning its S-dimensional coordinates $\Lambda(i,\cdot)$. As a result, each element of the S-dimensional embedding $\Lambda(i,\cdot)$ of the node i quantifies the proximity, in terms of Wasserstein distance on the graph, of its connectivity descriptor to the S graph structural patterns. The dimensionality of the embedding space S is a design choice and typically it depends on the number of clusters in the graph.

Optimization: We train node2coords in order to learn the graph representations M_S and Λ by minimizing a loss $\mathscr{L}(\hat{Z}_n, Z_n)$ between the node connectivity descriptors Z_n and their reconstruction as barycenters $\hat{Z}_n$. In order to ensure that the reconstruction of each node connectivity descriptor in Z_n is taken equally into account, we consider the normalized reconstruction loss:

$$\mathscr{L}(\hat{Z}_n, Z_n) = \frac{\|Z_n - \hat{Z}_n\|_F^2}{\|Z_n\|_F^2}. \tag{3.11}$$

It can be seen from Algorithm 2 and Eq. (3.10), (3.11) that $\mathscr{L}(\hat{Z}_n, Z_n)$ is differentiable with respect to Δ as well as with respect to M_S. Therefore, $\mathscr{L}(\hat{Z}_n, Z_n)$ is differentiable with respect to Δ and E. The minimization of the loss function is a non-convex problem:

$$\min_{E,\Delta} \frac{\|Z_n - \hat{Z}_n\|_F^2}{\|Z_n\|_F^2}, \tag{3.12}$$

that can be optimized with automatic differentiation [61] and stochastic gradient descent (SGD) [65]. The number of barycenters computed in parallel J, introduced in Section 3.3.1, determines the batch size used for training with SGD and it is a design choice. As the energy in Eq. (3.12) is non-convex, the graph patterns in M_S, and as a result, also the barycentric

coordinates Λ, will not be exactly the same for each run of node2coords. However, in every case, the graph patterns will be localized on the clusters of the graph. It can finally be noted that larger values of the entropy regularization ϵ constitute the energy in Eq. (3.12) less non-convex.

3.5 Experimental Results

3.5.1 Settings

In this section we evaluate the quality of the representations learned with node2coords for node classification tasks and examine their stability to perturbations of the graph structure. We compare the performance of our algorithm against the following unsupervised learning methods:

- Laplacian Eigenmaps (LE) [47]: A shallow-embedding method that finds S-dimensional node embeddings by keeping the eigenvectors of the graph Laplacian matrix that correspond to the S smallest eigenvalues. LE embeddings naturally emphasize the clusters in the graph.
- DeepWalk [52]: An algorithm that uses random walks on graphs to learn S-dimensional representations of nodes with a skip-gram model.
- node2vec [53]: A skip-gram method that uses biased random walks on graphs allowing for a trade-off between homophily and structural equivalence of the obtained node embeddings.
- SDNE [56]: An autoencoder that learns S-dimensional node embeddings at the ouput of the $N \times S$ linear layer of the encoder. SDNE embeddings preserve first-order and second-order node proximities.
- DVNE [22]: An autoencoder that learns S-dimensional Gaussian distributions in the Wasserstein space as the latent representation of the nodes. DVNE embeddings preserve the graph structure while simultaneously modelling the uncertainty of nodes.
- GAE [66]: An autoencoder that learns S-dimensional node embeddings by employing two graph convolutional layers in the encoder as $\mathrm{GCN}(X, A) = \bar{A}\mathrm{ReLU}(\bar{A}XW_0)W_1$ [67], where $\bar{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, and W_0, W_1 are trainable parameters. Further, X is the feature matrix of the nodes. In order to ensure a fair comparison with other methods, which do not use features, we replace the feature matrix X with the identity matrix, as proposed in [66].

The above methods were chosen in order to ensure comparison with benchmark methods as well as state-of-the-art methods for unsupervised learning of graph representations.

We consider the following datasets:

- Karate: The Zachary Karate network [68] is composed of $N = 34$ nodes. Each node is a member of a Karate university club, which is split into two communities.
- PolBooks: This dataset consists of a network of $N = 105$ books about US politics published around the time of the 2004 presidential election and sold by the online bookseller Amazon.com [69]. Edges between books represent frequent copurchasing of books by the same buyers. The books belong to one of three classes “liberal", “conservative", “neutral".
- Citeseer4: Citeseer [70] is a dataset that consists of a citation network extracted from the Citeseer digital library. Nodes are publications and an edge exists between two nodes if either publication has cited the other one. The publications belong to one of 6 classes, where each class corresponds to a research area. Citeseer4 is a network of $N = 1532$ nodes and corresponds to the giant component of the network obtained by the publications that belong to the 4 research areas of the Citeseer network.

3.5.2 Interpretation of node2coords on a Community Detection Task

We first evaluate the node embeddings learned by node2coords for the task of community detection on the Zachary Karate network. We also take the opportunity to explain in detail the representations learned with node2coords and build intuitions on how to select optimally its parameters.

We run node2coords for $C = D_{\tau=1}$ and $n = 1$ because the Karate network is a small graph and therefore 1-hop structural information is sufficient. We set $S = 2$ because there are two communities in the graph. For the remaining parameters, we set $\epsilon = 0.03$, $\rho = 0.05$. The choice of the value for the parameter ϵ determines how localized the graph structural patterns are and ρ controls the mass relaxation allowed. Furthermore, $L = 500$ iterations are sufficient for the barycenter computation to converge. The input in node2coords is the connectivity matrix $Z_{n=1}$ and the output is its reconstruction using barycenters $\hat{Z}_{n=1}$. We train using Adam [71] with learning rate $\mu = 0.01$.

The graph structural patterns in M_S are shown in Fig. (3.5). It can be seen that the low-dimensional space M_S comprises two very localized structural patterns which are placed on the two communities. The embeddings obtained with node2coords are shown in Fig. (3.3). The values of the barycentric coordinates λ_1, λ_2 of the nodes capture the proximity in terms of Wasserstein distance of the node connectivity descriptors relatively to the graph structural patterns of M_S. As an example, the connectivity descriptor of the node in Fig. (3.4) is approximated in the decoder as $\text{argmin}_{u \in \Sigma_N} \sum_{i=1}^{2} \lambda_i W_p^{\epsilon,\rho}(M_S(\cdot, i), u)$ where $M_S(\cdot, 1), M_S(\cdot, 2)$ are the two patterns shown in Fig. (3.5) and $\lambda_1 = 0.43, \lambda_2 = 0.57$ are the barycentric coordinates

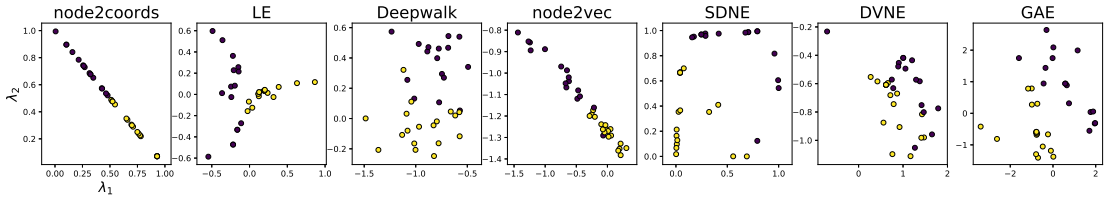


Figure 3.3 – Embeddings obtained for the Zachary Karate network with node2coords, LE, Deepwalk, node2vec, SDNE, DVNE and GAE. For node2coords the two axes correspond to the barycentric coordinates λ_1, λ_2. The embeddings of the other methods are not in a known coordinate system. The embeddings of the two communities are most clearly separated with node2coords.

learned for that node. Therefore, the nodes that are close to the first pattern in M_S have a large λ_1 and a small λ_2, as can be seen by the embeddings of the nodes in yellow in Fig. (3.3). The opposite is true for the nodes in the purple community.

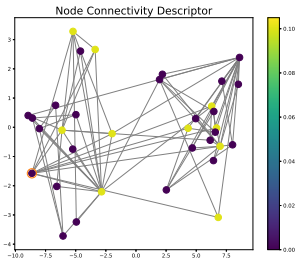


Figure 3.4 – Connectivity descriptor of the node highlighted with the orange circle.

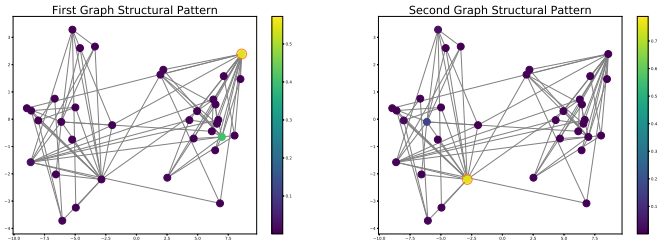


Figure 3.5 – Graph structural patterns learned in M_S for the Zachary Karate network. The colormap shows the range of intensities of the pattern on the graph. The patterns learned in M_S are placed on each one of the communities.

We also show in Fig. (3.3) the embeddings learned by Laplacian Eigenmaps, Deepwalk, node2vec, SDNE, DVNE and GAE for latent dimensionality $S = 2$ for the Zachary Karate Network. For node2vec, the parameters p, q are set to $p = 1, q = 0.5$, which were proposed as optimal in [53] for the task of community detection, where homophily among nodes is detected. For Deepwalk the number of walks is set to $\gamma = 10$, the walk length to $t = 10$ and the window size to $w = 3$. For SDNE, the regularization for the first order proximities is set to $\alpha = 0.16$, the L_2 norm regularizer to avoid overfitting equal to $\nu = 0.15$ and the reconstruction

penalization parameter to $\beta = 5$. For DVNE we set the size of the hidden layer to $h = 12$ and the parameter that controls the second-order proximity preservation to $\alpha_D = 1$. For GAE we set the dimensionality of the embeddings at the hidden layer of the encoder to $h = 32$. It can be seen that the embeddings obtained with all competitor unsupervised methods do not separate the two communities as clearly as node2coords. Further, the node embeddings of these methods lack interpretability as the value of the embedding of a node in each one of the two dimensions does not correspond to the proximity to a particular axis.

It can be observed that LE and node2coords discover the clusters in the graph. Specifically, LE, as explained in detail in [47], has a close connection to spectral clustering [104]. The embeddings obtained with LE, shown in Fig(3.3), correspond to the S eigenvectors f corresponding to the smallest, non-zero eigenvalues e of the generalized eigenvector problem $Lf = eDf$, where D is the diagonal matrix of node degrees and $L = D - W$ is the Laplacian matrix of the graph. Therefore, for the Zachary Karate Network of two clusters, the embedding of the i-th node obtained with LE corresponds to $[f_1(i), f_2(i)]$. Further, due to the use of the diffusion distance as a cost $C = D_{\tau=1}$, node2coords learns graph structural patterns in M_S that highlight the clusters of the graph, as shown in Fig. (3.5). The embeddings obtained with node2coords, shown in Fig(3.3), correspond to the Wasserstein distance of the node descriptors from those graph structural patterns. Therefore, although both LE and node2coords reveal the clusters of the graph, they lead to different node embeddings.

3.5.3 Stability to Perturbations

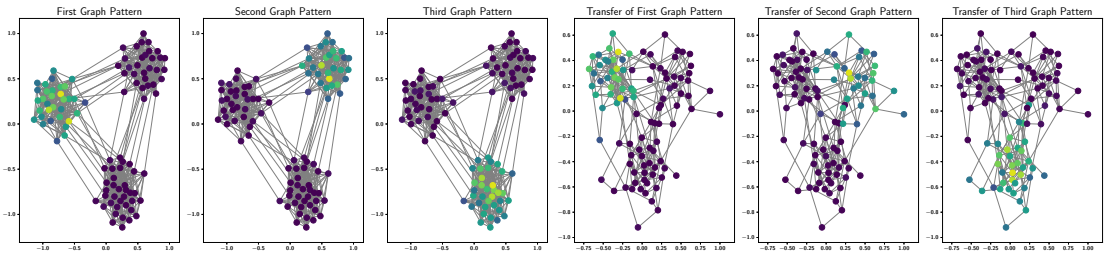
We now examine the stability of node2coords to perturbations of the graph structure. First, we consider a stochastic block model [72] graph G of $N = 100$ nodes with probability of connection within the community equal to $p = 0.4$ and probability of inter-community connection equal to $q = 0.01$. We then consider perturbed versions G' of the graph G by varying the probability p within the range $p' = \{0.15 : 0.05 : 0.40\}$. Therefore, the perturbation affects the number of edges of the graph, but the number of nodes remains constant.

We run node2coords with $n = 1$, $S = 3$, $\epsilon = 0.01$ and $\rho = 0.1$ and learn the space M_S and the barycentric coordinates Λ for the graph G. The graph structural patterns learned in M_S for the clean graph G are shown in Fig. (3.6a) and their transfer to the perturbed graph G' with $p' = 0.15$ in Fig. (3.6b). The graph structural patterns are less localized in this case compared to those in Fig. (3.5). This, as explained in Section 3.4, is due to the fact that the entropy regularization parameter ϵ used for the SBM graph is smaller than the one used for the Karate network. However the interpretation of the graph structural patterns remains the same. Specifically, it can be seen that the graph patterns in M_S identify the three communities and thus they remain meaningful even when the actual graph changes. As a result, the perturbed graphs can be embedded in the low-dimensional space M_S that was learned for the clean graph G. We confirm this intuition by evaluating the clustering result obtained using the node

embeddings Λ' of the perturbed graphs in the space M_S learned for the original graph G. For the perturbed graphs G' we only compute the barycentric coordinates Λ' of their nodes in the space M_S learned on G. We apply k-means clustering to the barycentric coordinates of the nodes Λ' with $k = 3$ and we compute the adjusted mutual information (AMI) and the normalized mutual information (NMI) [73] for the clustering result. The obtained AMI, NMI for the different perturbations are shown in Fig. (3.7). It can be seen that both the AMI and NMI are high even for large perturbations. Thus, we confirm that the perturbed graphs G' can be embedded in a meaningful way in the space M_S learned for the clean graph G.

We further evaluate the relative change $\frac{\|\Lambda-\Lambda'\|_F}{\|\Lambda'\|_F}$ in terms of Frobenius norm of the barycentric coordinates Λ' of the perturbed graphs G' in comparison to the barycentric coordinates Λ of the original graph G. In Fig. (3.8a) we show the relative change of the embeddings obtained as a function of the relative change of the probability of connection within the community $\frac{|p-p'|}{|p'|}$. Laplacian Eigenmaps, node2vec, Deepwalk, SDNE, DVNE and GAE do not learn a low-dimensional space as node2coords and, therefore, the only way to obtain the node embeddings of the perturbed graphs is by re-running the algorithms. It can be seen clearly that the embeddings obtained with node2coords are stable. DVNE produces also relatively stable embeddings. This is expected, as possible uncertainties of the node embeddings are accounted for through the variance of the Gaussian distribution. We notice also that the change in the node embeddings of SDNE seems to follow an increasing trend as the relative change of the probability of connection within the community increases. The relative change in the embeddings of node2coords seems to increase linearly with the relative change in the probability of intra-connection p. This is also clearly seen in Fig. (3.9) where we plot the node embeddings for node2coords. It can be seen that node embeddings obtained with node2coords change progressively as the value of the probability of connection p changes.

Furthermore, we consider the graph G of the PolBooks dataset and we create perturbed



(a) (b)

Figure 3.6 – (a) Structural graph patterns of M_S as learned for the graph G with $p = 0.4$. Each pattern identifies one of the communities. (b) Structural graph patterns of M_S learned for the graph G with $p = 0.4$ transferred to the perturbed graph G' with $p' = 0.15$. The graph structural patterns remain meaningful for the perturbed graph G' as they clearly indicate the three communities.

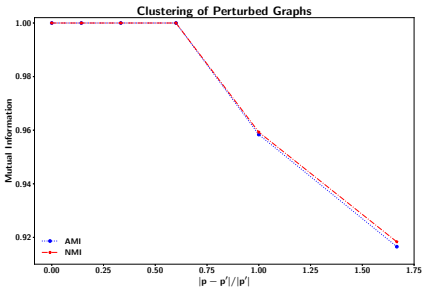
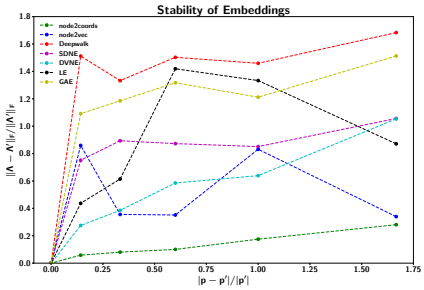
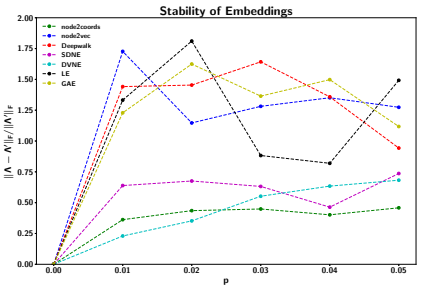


Figure 3.7 – Clustering of perturbed graphs. AMI and NMI scores as a function of the relative change of the probability of connection within the community $\frac{|p-p'|}{|p'|}$.



(a) Change of the embeddings $\frac{\|\Lambda-\Lambda'\|_F}{\|\Lambda'\|_F}$ as a function of the change of the intra-community probability of connection $\frac{|p-p'|}{|p'|}$ for the SBM graphs.



(b) Change of the embeddings $\frac{\|\Lambda-\Lambda'\|_F}{\|\Lambda'\|_F}$ as a function of the percentage of perturbed edges in the graph $p = \frac{|E_p|}{|E|+|E_p|}$ for the PolBooks dataset.

Figure 3.8 – Stability of node2coords embeddings.

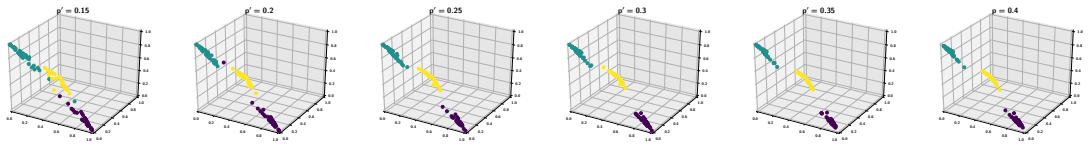


Figure 3.9 – Embeddings of node2coords of the perturbed graphs G' in the space M_S learned for the clean graph G.

graphs G' by randomly adding edges. We denote the number of added edges as $|E_p|$. The maximal number of edges that can be added to this network until it becomes fully connected is $|E_p|^{max} = N^2 - |E| = 10143$. We define the ratio of added edges as $r_p = \frac{|E_p|}{|E_p|^{max}}$ and generate the perturbed graphs G' by varying the ratio r_p in the range $r_p = \{0.01 : 0.01 : 0.05\}$. We run node2coords with $n = 1$, $S = 3$, $\epsilon = 0.01$ and $\rho = 0.1$ and learn the space M_S and the barycentric coordinates Λ for the graph G. Next, we compute the barycentric coordinates Λ' of the perturbed graphs G' in the space M_S and evaluate the relative change $\frac{\|\Lambda-\Lambda'\|_F}{\|\Lambda'\|_F}$ in terms of Frobenius norm of the barycentric coordinates Λ' of the perturbed graphs G' in

comparison to the barycentric coordinates Λ of the original graph G. In Fig. (3.8b) we show the relative change of the embeddings obtained as a function of the percentage of perturbed edges in the graph $p = \frac{|E_p|}{|E|+|E_p|}$. It can be seen that again node2coords provides the most stable node embeddings followed by DVNE. We notice also that SDNE produces relatively stable embeddings. Furthermore, we notice that for the smaller perturbations p, DVNE produces the most stable embeddings. Contrary to the perturbations generated for the SBM graphs, where only the probability of connection within the community is perturbed, the perturbations created in this experiment add randomly edges either within or between clusters. Therefore, in this case, the perturbed edges affect more strongly the clusters in the graph. These perturbations are more challenging for node2coords, since the geometry-aware cost relies on the diffusion distance, which naturally emphasizes the clusters in the graph. As the perturbation becomes stronger, the embeddings of DVNE become less stable than those of node2coords.

To conclude, we have shown experimentally that node2coords learns stable node embeddings. The advantage of the stability of the embeddings with node2coords is due to the fact that the low-dimensional space M_S permits a registration of the nodes in the case of perturbed graphs.

3.5.4 Node Classification

We now evaluate the features learned in an unsupervised manner with node2coords in the context of node classification tasks. The node embeddings learned by node2coords and the competitor methods are input to a one-vs-rest logistic regression classifier with L2 regularization.

We consider the same experimental set-up as in [53]. We consider train-test partitions of the data varying from 20% to 80%. For each partition we create 10 random splits of the data to train and test and provide results averaged over the 10 splits. The same splits are used for all methods. The F1 score of a class is the harmonic mean of precision and recall. The precision for a class is the number of true positives divided by the total number of elements labeled as belonging to the class and recall is the number of true positives divided by the total number of elements that belong to the class. We therefore compute the F1 score as:

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3.13}$$

with:

$$\text{precision} = \frac{TP}{TP + FP} \tag{3.14}$$

and

$$\text{recall} = \frac{TP}{TP+FN}, \tag{3.15}$$

where TP, FP, TN, FN stand for true positives, false positives, true negatives and false negatives, respectively.

For the evaluation of the classification results we compute the weighted Macro-F1 score, which calculates the F1 score for each label, and finds their weighted mean.

We present in Table (3.1) the Macro-F1 scores for the PolBooks dataset and for train-test splits varying from 20% to 80%. For node2coords and the algorithms against which we evaluate its performance, the dimensionality of the embedding space is considered to be equal to $S = 3$ as the network essentially has three clusters. It can be seen that node2coords provides the highest Macro-F1 score for most training ratios. Indeed, even for the small latent dimensionality of $S = 3$, it clearly identifies the small class “neutral". This can be clearly seen also from the graph structural patterns of M_S in Fig. (3.10). Specifically, DeepWalk, node2vec, SDNE and GAE make most of the classification errors for the small class “neutral", which leads to reduced Macro-F1 scores. This shows that the embedding dimensionality $S = 3$ is too small for these algorithms to properly capture the three clusters in the data, while it is sufficient for node2coords. DVNE consistenly has the second best performance after node2coords when the training ratio is more than 40%.

Table 3.1 – Macro-F1 score for node classification in PolBooks.

Train Ratio	20 %	30 %	40 %	50 %	60 %	70 %	80 %
LE	70.50	72.83	75.47	70.14	72.85	77.27	73.33
node2vec	72.48	75.52	74.77	70.80	73.61	80.76	**86.92**
DeepWalk	74.82	75.52	76.19	69.07	70.81	76.00	**86.92**
SDNE	64.48	65.40	68.65	70.65	70.81	72.22	79.16
DVNE	55.90	57.98	56.99	71.38	75.46	81.68	83.90
GAE	73.14	**79.58**	76.19	67.49	68.33	68.94	73.33
node2coords	**75.58**	73.26	**78.43**	**73.58**	**78.88**	**86.11**	**86.92**

We now examine how node2coords performs on larger datasets. We show in Table 3.2 the Macro-F1 scores for node classification on Citeseer4. The latent dimensionality for all methods is taken to be equal to $S = 4$. It can be seen again that node2coords provides the highest Macro-F1 scores for most train ratios. Therefore, node2coords is able to capture structural information for this larger graph. SDNE now has the second best performance, which shows that it is a method that scales well for large datasets. Also, node2vec performs consistently better than DeepWalk. This is due to the bias added to the random walks of node2vec, which

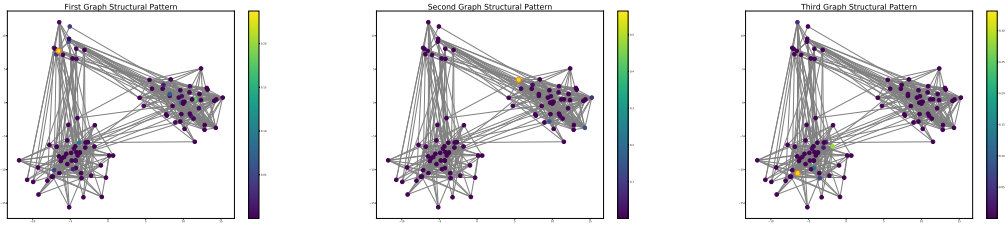


Figure 3.10 – Graph structural patterns learned for PolBooks. The node with the highest value of each pattern is highlighted with an orange circle. Each graph structural pattern indicates a cluster of the graph.

Table 3.2 – Macro-F1 score for node classification in Citeseer4.

Train Ratio	20 %	30 %	40 %	50 %	60 %	70 %	80 %
LE	30.78	32.47	31.81	32.87	34.09	34.57	36.63
node2vec	**68.56**	**69.97**	69.23	69.75	69.84	70.96	74.44
DeepWalk	57.56	59.61	59.01	59.18	59.10	60.69	60.27
SDNE	66.24	67.25	69.31	70.92	70.96	71.59	75.94
DVNE	38.48	40.60	40.24	40.63	40.86	39.47	41.45
GAE	61.44	63.86	63.34	64.01	65.46	65.33	67.00
node2coords	66.94	68.63	**70.58**	**72.14**	**74.12**	**75.97**	**78.80**

leads to embeddings that capture both homophily as well as structural similarities of nodes. The performance of DVNE drops significantly in this case. This is due to the low embedding dimensionality of $S = 4$. Specifically, as observed by the authors in [22], the quality of the embeddings of DVNE decreases abruptly when the embedding dimensionality drops below a threshold. For instance, they report that for a graph of $N = 2708$ nodes, the minimum embedding dimensionality is $S = 32$. Therefore, the embedding dimensionality of $S = 4$ is insufficient for Citeseer4, which is a network of $N = 1532$ nodes. The worst performing method is LE, which does not scale well to larger graphs. The drop in performance of Laplacian Eigenmaps for large graphs could be due to the fact that it only takes into account first order proximities of the nodes. This is also indicated by the fact that the performance of SDNE, which takes into account second-order as well as first-order node proximities, improves for large graphs. Further, we notice that the performance of GAE is worst than that of SDNE and node2vec. This is consistent with the results reported in [66] for the featureless case and highlights the fact that deep neural networks with graph convolutional filters are particularly relevant in the case where features are available. However, they do not seem to fully exploit the structural information of the graph.

3.5.5 Generalization to Unseen Nodes

In this experiment we evaluate the ability of node2coords to generalize to nodes that have not been seen during the representation learning process. In order to do so, we use only a downsampled version of the adjacency matrix of the PolBooks network during the representation learning process. Specifically, we use a randomly selected set of nodes for training and subsample the $N \times N$ adjacency matrix A to obtain the $N^{train} \times N^{train}$ training adjacency matrix A^{train}. From A^{train} we obtain the training connectivity matrix Z_n^{train}. With Z_n^{train} as the input to node2coords, we learn the $N^{train} \times S$ space M_S^{train} and the S-dimensional barycentric coordinates for each of the training nodes in that space. We eventually use the S-dimensional barycentric coordinates to train a one-vs-rest logistic regression classifier with l2 regularization.

As it has been shown above, the graph structural patterns learned with node2coords are sparse and they only have non-zero values on a small set of nodes within a given cluster. Therefore, we can upsample the patterns in M_S^{train} by zero padding in order to obtain the $N \times S$ space M_S. The graph structural patterns in M_S obtained this way are meaningful as they indicate the clusters in the graph. We validate the quality of the patterns in M_S by evaluating the classification performance of the unseen nodes' coordinates in the space defined by M_S. Specifically, we compute the barycentric coordinates of the unseen, test nodes in the space M_S using the barycentric decoder of node2coords with fixed input M_S. We predict their class labels using the trained logistic regression classifier and evaluate the classification accuracy. We consider downsampling partitions ranging from 50 % to 90 %. For each partition we create 5 random splits of the data to train and test and provide results averaged over the 5 splits.

In Table (3.3) we show the classification accuracy for node2coords for downsampling ratios ranging from 50 % to 90 % (node2coords-DS) as well as the classification accuracy for the set-up of Section 3.5.4 (node2coords). We can see that the algorithm generalizes well to nodes that were completely unseen while learning the representation M_S with node2coords. When only 50 % of the nodes are kept in A^{train} the classification accuracy for the nodes unseen during learning of M_S is 76.15 %.

Table 3.3 – Accuracy of node classification in PolBooks.

Train Ratio	50 %	60 %	70 %	80 %	90 %
node2coords	86.79	90.47	93.75	95.23	100.00
node2coords-DS	76.15	76.19	78.06	83.80	80.00

Node2coords generalizes well to unseen nodes because the patterns learned for the downsampled graph capture the most important structural information which, in this case, corresponds to the clusters. The ability of node2coords to learn such a meaningful low-dimensional representation of the graph, given only partial information of the graph

structure, is unique to node2coords and cannot be reproduced by other methods for graph representation learning that only leverage structural information but not node features.

3.5.6 Parameter Sensitivity

We now examine the sensitivity of the quality of the node embeddings learned with node2coords with respect to the entropy regularization parameter ϵ and the relaxation parameter for the mass preservation constraints ρ.

We first investigate the sensitivity of node2coords with respect to the parameter ϵ. We set $\rho = 0.1$ and vary ϵ in the range $\epsilon = \{0.01 : 0.01 : 0.09\}$. The classification accuracy as a function of ϵ is shown in Fig. (3.11). It can be seen that the classification accuracy is relatively stable for ϵ in the range $\epsilon = \{0.01 : 0.01 : 0.05\}$ and drops for values of $\epsilon > 0.05$. The reason for this drop in the perfromance for larger values of ϵ is directly linked with the quality of the graph structural patterns. Specifically, as the entropy regularization parameter ϵ increases, the graph patterns are forced to be more localized. Significant increase of the regularization parameter ϵ forces the graph patterns to be Dirac δ functions, equal to 1 on a node and 0 everywhere else. However, as can be seen in Fig. (3.10) the optimal graph structural patterns are localized on a specific cluster and not on a single node. In the case where the graph structural patterns are Dirac δ functions, the barycentric coordinates will quantify the proximity with respect to S nodes and not the proximity with respect to the S clusters, as desired.

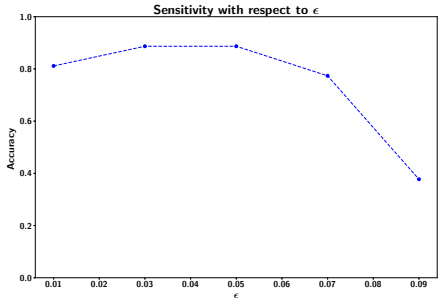


Figure 3.11 – Sensitivity of the classification accuracy on the PolBooks dataset with respect to the entropy regularization parameter ϵ.

Further, we mention that the performance of node2coords is not significantly affected by the value of the parameter ρ. Specifically, even if ρ is set to a smaller value than necessary, or equivalently if the mass preservation constraints are relaxed more than necessary for the optimal reconstruction of the node connectivity descriptors, the mass of the barycenters will converge to that needed for the optimal reconstruction. We note that very large values of ρ take us away from the ubalanced barycenter computation. In that case, there will be a drop in performance because the mass preservation constraints may penalize the optimal reconstruction of the node connectivity descriptors obtained with the Wasserstein barycenters. This is the reason why we have chosen to compute unbalanced barycenters in the decoder of

node2coords.

3.5.7 Study of the Effect of the Barycentric Layer

In this section we study the effect of using our proposed barycentric layer compared to a simple linear layer in the decoder. In order to do so, we compare the performance of node2coords to SDNE, which is composed of a linear layer in the encoder and the decoder, when both its regularization parameters are set to $\alpha = 0$ and $\nu = 0$. Furthermore, we provide as the input to both node2coords and SDNE the node connectivity descriptors. We show in Tables 3.4, 3.5 the Macro-F1 scores obtained on the PolBooks and the Citeseer4 dataset for SDNE (with no regularizers) and node2coords. It can be seen that node2coords significantly outperforms SDNE with no regularizers. We have therefore illustrated the benefit of the barycentric layer compared to a linear layer.

Table 3.4 – Effect of barycentric layer. Macro-F1 scores for node classification in PolBooks.

Train Ratio	20 %	30 %	40 %	50 %	60 %	70 %	80 %
SDNE (no reg.)	66.27	61.42	62.22	61.27	63.81	64.29	64.91
node2coords	**75.58**	**73.26**	**78.43**	**73.58**	**78.88**	**86.11**	**86.92**

Table 3.5 – Effect of barycentric layer. Macro-F1 scores for node classification in Citeseer4.

Train Ratio	20 %	30 %	40 %	50 %	60 %	70 %	80 %
SDNE (no reg.)	63.69	62.69	63.99	64.35	63.38	63.65	63.08
node2coords	**66.94**	**68.63**	**70.58**	**72.14**	**74.12**	**75.97**	**78.80**

3.6 Conclusion

In this work we proposed node2coords, an autoencoder architecture with a novel Wasserstein barycentric decoder that learns low-dimensional graph representations without supervision. The proposed algorithm learns simultaneously i) a low dimensional space and ii) node embeddings that correspond to coordinates in that space. The low-dimensional space is defined by a small set of graph patterns that capture the most relevant structural information of the graph. The values of a node's embedding in that space can be interpreted as the proximity of its local connectivity to the corresponding graph patterns in terms of Wasserstein distance on the graph.

We demonstrated how the low-dimensional space of node2coords can be used to obtain significantly more stable embeddings for graphs that have undergone small perturbations, compared to other methods. Furthermore, we showed that the node embeddings of node2coords provide competitive or better results than those obtained with state-of-the-art

methods for node classification tasks on real datasets. Finally, we confirm experimentally the ability to generalize to nodes that were unseen during the representation learning process.

4 Pooling Graph Representations with Wasserstein Gradient Flows

4.1 Introduction

In the previous chapter, we focused on the case where we learn representations for a fixed graph. Specifically, by exploiting the structure of a graph, we learned embeddings of its nodes in order to perform node classification and community detection. However, in machine learning problems, such as graph classification or graph regression, each datapoint corresponds to a different graph. Examples of such problems arise in the fields of computational biology, drug discovery and social network analysis, among others. As an example, a task of interest is the prediction of whether a protein structure is an enzyme or not. In that case, proteins are represented as graphs with nodes corresponding to amino-acids and edges capturing the spatial proximity of the amino-acids. Similarly, molecules can be represented as graphs where nodes are atoms and edges the chemical bonds between them. The problem of interest in that case may be the classification of a molecule as active or inactive against cancer cells.

Such tasks are tackled by models that operate in an inductive setting. In this setting, the goal of graph representation learning algorithms is to use a set of k training graphs $G_1, \dots G_k$ in order to learn a mapping that can generalize to unseen test graphs $G_{k+1}, \dots G_{k+l}$. Naturally, the graphs can be of varying size. For instance, in the example of molecule classification discussed above, each molecule may be composed of a different number of atoms. As a result, a relevant problem that arises in GNN architectures in this context, is to find the optimal way to pool the graph representations of varying size to a representation of fixed size that can be multiplied by the weights of the classifier or regressor that is driving the representation learning process. This operation is commonly referred to as global graph pooling.

Graph pooling methods proposed fall into two categories: node selection methods and node clustering methods. Node selection methods aim to pool graph representations by selecting the representations of the most important nodes of the graph, according to some criterion. An

important disadvantage of these methods is that they discard the information that is captured in the representations of the least important nodes. Node clustering methods propose to pool graph representations by learning node assignment matrices. The motivation of these methods is to learn what constitutes nodes to be similar, so that the representations of similar nodes can be linearly combined, as dictated by the assignment matrices. Node clustering methods don't discard information, but aggregate the representations of similar nodes. However they still offer no guarantee with regards to the similarity of the graph representation and its pooled version.

In this work, we introduce a new type of pooling method, that addresses this limitation. We propose to explicitly preserve the statistics of the graph representation by minimizing the entropy-regularized Wasserstein distance between itself and its pooled version. Our pooling method minimizes this distance by performing a Wasserstein gradient flow with respect to the pooled graph representation. Therefore, we term our proposed pooling method FlowPool. At each step of the flow, the energy of the Wasserstein distance between the graph representation and its pooled counterpart is computed and the pooled representation is moved closer to the original graph representation according to the gradient of that energy. As a result, the pooled graph representation, obtained with FlowPool, combines the representations of the nodes using the optimal couplings that minimize the Wasserstein distance along the steps of the flow.

Our contributions in this work are threefold:

- We propose a global pooling framework for graphs that optimally preserves the statistics in the representation space.
- We propose a versatile implementation of this pooling method using implicit differentiation [74] that can take into account any underlying geometry and that enables its integration in end-to-end deep learning architectures for graphs.
- We analyse the proposed pooling method for the case where the geometry considered is that determined by the squared Euclidean distance in the feature space and provide promising results for graph classification on real data.

The structure of this chapter is as follows. First, in Section 4.2 we review the related work. After introducing the correspondence of a graph representation to a probability measure in Section 4.3, we propose FlowPool as the minimization of the Wasserstein distance between graph representations in Section 4.4. In Section 4.5 we explain our versatile implementation based on implicit automatic differentiation that can be used with any ground cost. In Section 4.6 we show that our method is invariant to permutations. In Section 4.7 we discuss how we can backpropagate through FlowPool and integrate it in end-to-end GNN architectures. In Section 4.8.1 we perform an experimental analysis for a simplified set-up of FlowPool in order

to show its dependence on parameters and build intuitions and in Section 4.8.2 we provide preliminary results on graph classification when FlowPool is incorporated in an end-to-end deep learning architecture. We provide direction for future work in Section 4.9 and conclude in Section 4.10.

4.2 Related Work

As mentioned before, existing pooling methods for graphs can be grouped into two categories, namely node selection methods and node clustering methods. In this section we review briefly representative methods from each of these two categories.

Node selection methods pool a graph representation by keeping only the representations of the K most important nodes in the graph. The first node selection method proposed is SortPool [75], which extends the idea of the Weisfeiler-Lehman graph kernels [25] to sort nodes based on their color, to sorting nodes based on their GCN feature vector. Once the nodes have been sorted, only the K most important ones are kept. Sortpool yields the same representation for isomorphic graphs. A different approach is followed by TopKPool [76]. TopKPool proposes to learn from the data a vector of parameters, such that only the K node representations, whose inner product with the trainable vector is maximal, are kept. TopKPool does not explicitly take into account the graph structure. This limitation is addressed by SAGPool [77]. SAGPool employs a trainable graph convolutional layer in order to obtain an attention score for each node in the graph, and keeps only the K nodes that correspond to the highest attention scores.

Node selection methods lead to loss of information due to the discarded features of the nodes that are not selected. Node clustering methods aim to alleviate this problem by finding the optimal way to aggregate the representations of all N nodes in a graph. This is achieved by clustering the N nodes into M clusters in order to obtain an $N \times M$ assignment matrix S. As a result, given a graph of N nodes with adjacency matrix A and a $d \times N$ graph representation Y, the representation output by node clustering pooling methods is equal to $X = YS$. Further, the assignment matrix S provides a way to obtain the adjacency matrix A_c of the pooled graph as $A_c = S^\top AS$. All proposed node clustering methods learn the assignment matrices from the data in order to achieve statistical strength. We provide a summary of existing node clustering pooling methods below.

The first differentiable graph pooling method that proposes to learn assignment matrices is DiffPool [78]. DiffPool proposes to parametrize node assignment matrices with a graph convolutional filter followed by a softmax activation. The parameters of the GCN filter are learned from the data during training. Therefore, DiffPool's assignment matrices learn which nodes should be grouped together, based on the features and the graph structure. Building on DiffPool's concept of parametrizing assignment matrices with GCNs, StructPool [79] proposes to condition the cluster assignment of each node on the cluster assignments of other nodes. As

a result, the graph pooling problem is framed as a structured prediction problem by employing conditional random fields to capture the relationships among the assignments of the nodes. On a different line, HaarPool [80] employs the Haar basis [81], [82] matrix to obtain a mapping of the nodes in the graph to the nodes of the pooled graph. The graph is pooled by keeping the basis vectors that correspond to the low frequencies, thus maintaining the coarse structural information, and discarding the ones that correspond to the high frequencies, thus dismissing fine structural information. In order to leverage the node features as well as the spectral information of the graph structure, MinCutPool [83] proposes to parametrize node assignment matrices using multi-layer perceptrons and adds a term at the objective function which is a relaxation of the normalized minCUT problem [84].

Graph pooling is employed in deep networks for graph representation learning in two different contexts; hierarchical pooling and global pooling. Hierarchical pooling aims to generalize the pooling operation, as performed in convolutional networks for regular grids, and aggregate feature information over local patches in order to achieve locality-preserving representations and invariance to small deformations. Global pooling aims to learn fixed-size representations from graph representations of varying size. In a recent study [85] it is demonstrated that certain graph pooling methods, when employed for hierarchical pooling, do not enhance the graph representation learning process, but rather smooth the features of the nodes in the pooled graph representation. Although in this work we focus on global graph pooling, we mention that our proposed pooling method can also be used for hierarchical pooling and that it could help alleviate the smoothing problem because of the explicit objective of preserving the statistics. In that context, the coarsened graph adjacency matrix A_c could be obtained through the optimal couplings.

Further, we mention that optimal transport-based methods have been recently proposed in order to obtain fixed-size representations. First, the work in [86] proposes a framework that uses a mapping to obtain a linear approximation of the Wasserstein-2 distance. In order to obtain this mapping, the barycentric projection [87] from a reference to each graph representation is needed. The optimal transport plans, needed for the barycentric projection map calculation, are obtained by solving the Kantorovich problem of Eq. (2.7) with linear solvers [88]. Once the graph representations of fixed size are obtained, they are used to perform various graph prediction tasks. As this framework is not differentiable, it cannot be integrated in end-to-end architectures for graph representation learning and only non-parametric graph representations are considered. Also, its performance relies heavily on the choice of the values of the representation of the reference, which must be numerically close to those of the graph representations. Second, in [89] the authors address the problem of finding fixed-size representations of sets of features of varying size and, possibly, in the regime where labeled data are scarse. They introduce an embedding (OTKE) that combines kernel methods [90] and optimal transport. They propose to embed the feature representations of a set to a reproducing kernel Hilbert space and subsequently pool the obtained embedding using the

optimal transport plan between the kernel embedding and a trainable reference. Furthermore, the relation of the proposed OTKE to attention mechanisms is discussed and its performance is validated on biological sequence classification and natural language processing tasks.

4.3 Graph Representations as Probability Measures

In order to define a Wasserstein distance between graph representations, we first explain how we correspond the representation of a graph to a probability measure. Given a graph G of N nodes, its representation with d features is a $d \times N$ matrix $Y = [y_1, \dots y_N]$ where y_j is the $d \times 1$ dimensional representation of the j-th node. We propose to describe the representation of the graph G as a probability measure:

$$\nu = \sum_{j=1}^{N} b_j \delta_{y_j}, \tag{4.1}$$

where $b \in \Sigma_N$ is a histogram. The value of b_j captures the significance of the representation of node j. In the case where all nodes are of equal importance we may consider $b_j = \frac{1}{N}, \forall j$. On the contrary, we can consider a non-uniform distribution on the nodes in order to account for some uncertainty in the node representation. For instance, in a social network, we may chose to use higher weights for nodes that correspond to users that have been members of the network for more that one year and whose features decribe them well, and lower weights for users that have just joined the social network.

In Fig. (4.1) we provide an example. We consider a graph G of $N = 7$ nodes with representation $Y \in \mathbb{R}^{2\times 7}$. The 2×1 dimensional feature representation of node j corresponds to $y_j = [Y_{1,j}; Y_{2,j}]$. We show on the left of Fig. (4.1) the graph G, where each node is annotated with its representation. We correspond to the representation of the graph G the probability measure $\nu = \sum_{j=1}^{7} b_j \delta_{y_j}$. Assuming that all nodes have equal importance, the weights of the measure ν are equal to $b_j = \frac{1}{7}, \forall j$. The positions of mass of the probability measure ν are determined by the representations of the nodes y_j. On the right of Fig. (4.1) we plot the measure ν in the 2-dimensional Euclidean space.

4.4 FlowPool

The goal of global graph pooling is to transform a representation of a graph of N nodes in a d-dimensional feature space to a fixed size representation of M nodes with the same feature dimension d. Let $\nu = \sum_{j=1}^{N} b_j \delta_{y_j}$ and $\mu = \sum_{i=1}^{M} a_i \delta_{x_i}$ be the probability measures that correspond to the graph representation $Y \in \mathbb{R}^{d\times N}$ and its pooled counterpart $X \in \mathbb{R}^{d\times M}$.

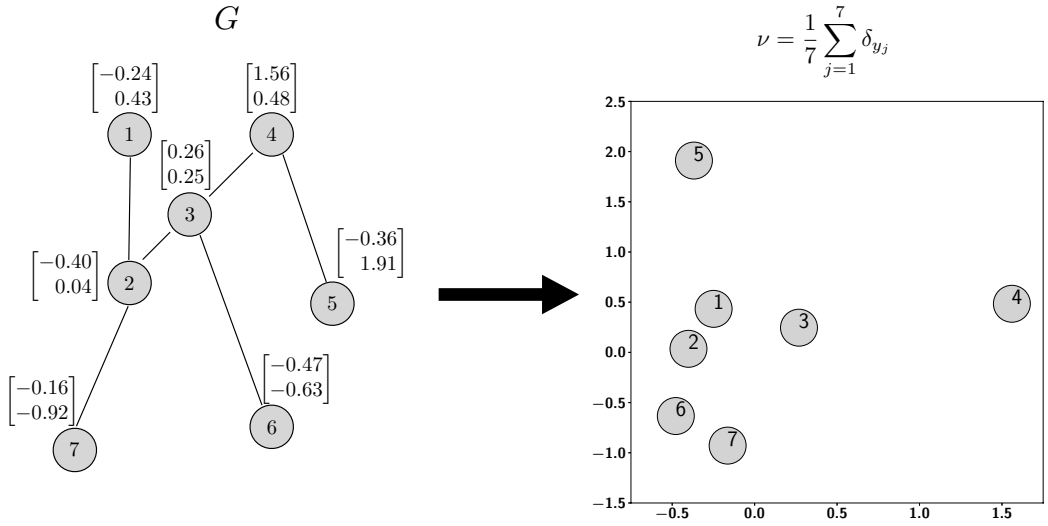


Figure 4.1 – Corresponding a graph G to a probability measure ν. On the left we show a graph G of $N = 7$ nodes with its representation Y. The j-th node is annotated with its representation $y_j = [Y_{1,j}; Y_{2,j}]$. For instance, the representation of node 2 is $y_2 = [-0.40; 0.04]$. By assuming that all the nodes in graph G are equally important, we correspond the graph representation to the probability measure $\nu = \frac{1}{7}\sum_{j=1}^{7}\delta_{y_j}$. The positions of mass of the measure are determined by the node representations.

FlowPool computes the pooled representation X by solving:

$$\min_{X} L^{\epsilon}_{C(X,Y)}(a,b), \tag{4.2}$$

where $L^{\epsilon}_{C(X,Y)}(a,b)$ is the entropy regularized Wasserstein distance defined in Eq.(2.11) for a cost $C = D^p$, with D a distance metric on the graph representation space $\Omega = \mathbb{R}^d$. We denote the cost of the mass transportation as $C(X,Y)$ in order to highlight the dependency of the cost C on X, Y. From now on, we assume that the nodes are of equal importance in all cases, so that $a = \frac{1}{M}\mathbb{1}_M$ and $b = \frac{1}{N}\mathbb{1}_N$, and we refer to $L^{\epsilon}_{C(X,Y)}(a,b)$ as the Wasserstein distance between the representations X, Y.

As discussed in Section 2.3.1, the objective function in Eq. (4.2) is differentiable with respect to X. Therefore, using the entropic regularization introduced in [13] we can compute the pooled graph representation by solving the problem in Eq. (4.2) with a gradient-based optimization method. Thus, our proposed pooling method is cast to a gradient flow. By denoting as $X^{(0)}$ the initialization for the pooled graph representation and as $X^{(L)}$ the obtained pooled representation after L iterations of the gradient-based method that minimizes Eq. (4.2), FlowPool can be thought of as demonstrated in Fig. (4.2). Specifically,

given an initialization $X^{(0)} \in \mathbb{R}^{d\times M}$, the FlowPool method summarizes any graph representation $Y \in \mathbb{R}^{d\times N}$, by performing L steps of a gradient-based method that minimizes the energy $L^{\epsilon}_{C(X,Y)}(a,b)$. The returned pooled representation $X^{(L)}$ is the $d \times M$ representation that has minimal Wasserstein distance from the $d \times N$ input representation Y.

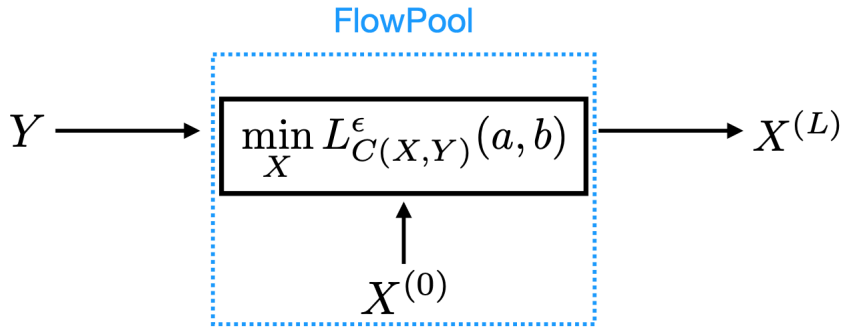


Figure 4.2 – The FlowPool method for pooling graph representations. The input is a graph representation $Y \in \mathbb{R}^{d\times N}$, where N can admit any value. The output $X^{(L)}$ is a $\mathbb{R}^{d\times M}$ representation, with M fixed, such that $L^{\epsilon}_{C(X^{(L)},Y)}(a,b)$ is minimal.

4.5 Implementation of FlowPool

At each step of the gradient-based optimization of X, we need to compute the gradient with respect to X of the energy $L^{\epsilon}_{C(X,Y)}(a,b)$ for the current X. One possible way to compute this gradient is to back-propagate through the operations of the Sinkhorn iterations. This is a similar approach to the way differentiation is handled in Chapter 3. A more efficient approach to compute $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ is by using the implicit function theorem [91]. The implicit differentiation of the entropy-regularized optimal transport problem is particularly more efficient in terms of memory requirements as not all intermediate calculations of the Sinkhorn iterations need to be stored in memory for the backpropagation.

Implicit Function Theorem

The implicit function theorem [91] states that, given a continuously differentiable function F, and a point $(x^*(\theta),\theta)$ such that $F(x^*(\theta),\theta) = 0$, if the Jacobian $\partial_1 F(x^*(\theta),\theta)$ is invertible, the variables $x(\theta)$ are differentiable functions of θ in some neighborhood of the point $(x^*(\theta),\theta)$.

By applying the chain rule, we obtain:

$$\begin{aligned} &\partial_1 F(x^*(\theta),\theta)\partial x^*(\theta) + \partial_2 F(x^*(\theta),\theta) = 0 \Leftrightarrow \\ &-\partial_1 F(x^*(\theta),\theta)\partial x^*(\theta) = \partial_2 F(x^*(\theta),\theta), \end{aligned} \tag{4.3}$$

where by ∂_1, ∂_2 we denote the Jacobian with respect to the first and the second argument, accordingly. As a result, the Jacobian $\partial x^*(\theta)$ can be obtained by solving the linear system in Eq. (4.3).

The recent work in [74] proposes to use the implicit function theorem in order to perform automatic differentiation of optimization problems. In that context, θ stands for the inputs of the optimization problem, $x^*(\theta)$ for the optimal solution and the function F captures the optimality conditions. Thus, the implicit function theorem offers the possibility to obtain the Jacobian of the optimal solution with respect to the inputs of the optimization problem.

Implicit Differentiation of the Entropy Regularized Optimal Transport Problem

In this work we use the `OTT` toolbox [92], which offers an implementation of the implicit differentiation of the entropy-regularized optimal transport problem [93]. We outline briefly below its implementation and how we employ it to compute the gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ in FlowPool.

By considering the dual formulation in Eq. (2.19), the computation of $L^{\epsilon}_{C(X,Y)}(a,b)$ consists in computing the optimal dual potentials f^*, g^* and the entropy regularized OT cost from these potentials using Eq. (2.19). Therefore, from the chain rule, we can obtain the gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ as:

$$\nabla_X L^{\epsilon}_{C(X,Y)}(a,b) = \partial[f^*(X), g^*(X)]^\top \nabla_{[f^*(X),g^*(X)]} L^{\epsilon}_{C(X,Y)}(a,b), \tag{4.4}$$

where the gradient $\nabla_{[f^*(X),g^*(X)]} L^{\epsilon}_{C(X,Y)}(a,b)$ captures how changes in the optimal potentials $f^*(X), g^*(X)$ affect the entropy-regularized optimal transport cost and the transposed Jacobian $\partial[f^*(X), g^*(X)]^\top$ shows how changes in the positions X affect the optimal potentials $f^*(X)$ and $g^*(X)$.

The Jacobian $\partial[f^*(X), g^*(X)]$ can be computed using the implicit function theorem. As discussed in Section 2.3.1, the iterations in Eq. (2.23) are a block-coordinate ascent on the dual problem. As a result, the optimality conditions consist in cancelled gradients of the energy $\mathscr{E}(f,g) = \langle f, a\rangle + \langle g, b\rangle - \epsilon\langle e^{\frac{f}{\epsilon}}, Ke^{\frac{g}{\epsilon}}\rangle$ with respect to f and g. Therefore, by considering the

continuously differentiable function F as:

$$F([f(X), g(X)], X) = \begin{bmatrix} \nabla|_f \mathscr{E}(f, g) \\ \nabla|_g \mathscr{E}(f, g) \end{bmatrix}, \tag{4.5}$$

at optimality it holds that:

$$F([f^*(X), g^*(X)], X) = \begin{bmatrix} 0_M \\ 0_N \end{bmatrix}. \tag{4.6}$$

The implicit function theorem states that, provided that the Jacobian $\partial_1 F([f^*(X), g^*(X)], X)$ is invertible, we can obtain the Jacobian $\partial[f^*(X), g^*(X)]$ by solving a linear system, such as the one in Eq. (4.3). From Eq. (4.5), we can obtain $\partial_1 F([f(X), g(X)], X)$ as:

$$\partial_1 F([f(X), g(X)], X) = \begin{bmatrix} \partial_f \nabla_f \mathscr{E}(f, g) & \partial_g \nabla_f \mathscr{E}(f, g) \\ \partial_f \nabla_g \mathscr{E}(f, g) & \partial_g \nabla_g \mathscr{E}(f, g) \end{bmatrix}. \tag{4.7}$$

By evaluating the derivatives in Eq. (4.7) at the optimal potentials f^* and g^* it follows that:

$$-\partial_1 F([f^*(X), g^*(X)], X) = \frac{1}{\epsilon} \begin{bmatrix} \mathrm{diag}(a) & P^* \\ P^{*\top} & \mathrm{diag}(b) \end{bmatrix}, \tag{4.8}$$

where P^* is the optimal coupling obtained for the transportation problem and $\mathrm{diag}(a), \mathrm{diag}(b)$ are diagonal matrices, with elements in the diagonal corresponding to the marginals a and b, respectively.

The Jacobian $\partial_2 F([f^*(X), g^*(X)], X)$ can be computed with automatic differentiation. Due to the block structure of the Jacobian in Eq. (4.8), the linear system in Eq. (4.3) can be solved with Schur's complement [94]. The Schur complement that is inverted depends on which of the M or N is smaller. In either case, the Schur complement is rank deficient, with a 0 eigenvalue for the vector of ones. Therefore, a ridge kernel regularization is added that enforces solutions to have zero sum. Further, when two nodes in the graph representation Y have very similar feature representations, two columns of the cost C are numerically close and therefore two columns of the optimal coupling P^* are colinear. In order to deal with the rank deficiency of the Schur complement in that case, a ridge identity regularization is added. The linear system is solved with the conjugate gradient method [95]. Having solved the linear system with respect to $\partial[f^*(X), g^*(X)]$, and using the chain rule, we can obtain the desired gradient as in Eq. (4.4).

We show in Algorithm 3, the implementation of FlowPool for L iterations of the gradient flow. In line 3 we compute the optimal potentials f^*, g^* using block coordinate ascent, as described in Chapter 2. We use the implicit differentiation in line 5 to obtain the gradient as in Eq.(4.4) and update the pooled representation using this gradient in line 7. The geometry is captured by the cost function $C(X,Y)$, which uses the input graph representation Y and its pooled counterpart $X^{(l)}$ at the l-th iteration, to return the cost matrix $C^{(l)}$. We denote with τ the step size used for the gradient flow.

Algorithm 3 FlowPool

Input: $Y; X^{(0)}$

1: **for** $l = 0$ to $L-1$ **do**
2: Solve Sinkhorn
3: $f^*, g^* = \text{Sinkhorn}\big(a, b, C^{(l)} = C(X^{(l)}, Y), \epsilon\big)$
4: Compute gradient with implicit function theorem
5: $\nabla_X L^{\epsilon}_{C(X^{(l)},Y)}(a,b) = \text{implicit\_diff}\big(f^*, g^*, a, b, X^{(l)}, Y, \epsilon\big)$
6: Update pooled graph representation:
7: $X^{(l+1)} = \text{update\_step}\Big(X^{(l)}, \nabla_X L^{\epsilon}_{C(X^{(l)},Y)}(a,b), \tau\Big)$
8: **end for**
9: **return** $X^{(L)}$

4.6 Permutation Invariance

We now show that the proposed pooling method is permutation invariant. This is a particularly relevant property when performing global pooling of graph representations. If FlowPool is preceded by permutation equivariant message passing layers [4], the pooled graph representation is permutation invariant. This means that the predictions made for a given graph are independent of the ordering of its nodes.

4.6.1 Computation of $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$

Although the gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ is computed in an automatic way in FlowPool using implicit differentiation, as outlined in Section 4.5, we demonstrate now how it can be computed analytically in order to prove that FlowPool is invariant to permutations. For this derivation, we consider the primal formulation of the entropy-regularized problem which, as explained in Chapter 2, is equivalent to the dual formulation. We remind that in the primal formulation, the optimization variable is the coupling P, and that the optimal coupling can be obtained from the optimal potentials f^*, g^* as:

$$P^* = e^{\frac{f^* \mathbb{1}_m^\top + \mathbb{1}_n g^{*\top} - C}{\epsilon}}. \tag{4.9}$$

Using the primal formulation, the gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ can be expressed as:

$$\nabla_X L^{\epsilon}_{C(X,Y)}(a,b) = [\partial_X C(X,Y)]^{\top} \nabla_C L^{\epsilon}_{C(X,Y)}(a,b), \tag{4.10}$$

where $\nabla_C L^{\epsilon}_{C(X,Y)}(a,b)$ shows how changes in the cost matrix C affect the entropy-regularized cost $L^{\epsilon}_{C(X,Y)}(a,b)$ and the transposed Jacobian $[\partial_X C(X,Y)]^{\top}$ shows how changes in the positions X affect the cost matrix C. The computations in Eq. (4.4) and Eq. (4.10) are equivalent. In Eq. (4.10) however, the only term that depends on the solution of the optimal transport problem is $\nabla_C L^{\epsilon}_{C(X,Y)}(a,b)$. Therefore, it is more intuitive to study the differentiation with respect to X using the primal formulation.

By observing Eq. (2.11) one can notice that in the case of FlowPool the objective function that is being minimized is a function of two variables, the cost C and the coupling P. As a result, we can re-express Eq. (2.11) as

$$L^{\epsilon}_{C}(a,b) = \min_{P \in U(a,b)} Q(C,P), \tag{4.11}$$

where:

$$Q(C,P) = \langle C,P \rangle - \epsilon H(P). \tag{4.12}$$

Therefore, the optimal coupling P^* is obtained by:

$$P^*(C) = \operatorname*{argmin}_{P \in U(a,b)} Q(C,P), \tag{4.13}$$

where we denote the coupling as $P^*(C)$ in order to highlight its dependency on the cost C. This dependency is a direct consequence of the factorization in Eq. (2.15), the Sinkhorn iterations in Eq. (2.18) and the expression for the Gibbs kernel in Eq. (2.16).

From Eq. (4.11), (4.12), (4.13) we obtain that:

$$L^{\epsilon}_{C(X,Y)}(a,b) = Q(C,P^*(C)). \tag{4.14}$$

As a result, from the chain rule it follows that:

$$\nabla_C L^{\epsilon}_{C(X,Y)}(a,b) = \nabla_1 Q(C,P^*(C)) + [\partial_C P^*(C)]^{\top} \nabla_2 Q(C,P^*(C)), \tag{4.15}$$

where we denote by ∇_1 and ∇_2 the gradient with respect to the first and the second argument respectively.

Due to the envelope theorem [96], [97], we can approximate $Q(C, P^*(C))$ with a function $G(C) = Q(C, P^*)$, where P^* is assumed to be constant and independent of C. Therefore, we obtain:

$$\nabla_C L^{\varepsilon}_{C(X,Y)}(a,b) = \nabla G(C) = \nabla_1 Q(C, P^*) = P^*. \tag{4.16}$$

From Eq. (4.10), (4.16) it follows that:

$$\nabla_X L^{\varepsilon}_C(a,b) = \Big(\sum_{j=1}^{N} P^*_{i,j} \nabla_1 C(x_i, y_j) \Big)_{i=1}^{M}. \tag{4.17}$$

4.6.2 Proof of Permutation Invariance

We consider a graph representation $Y_1 \in \mathbb{R}^{d \times N}$ and its column-wise, or equivalently node-wise, permutation according to an $N \times N$ permutation matrix $\mathscr{P}_f$:

$$Y_2 = Y_1 \mathscr{P}_f. \tag{4.18}$$

We will show that FlowPool(Y_1) = FlowPool(Y_2).

OT cost

Let $X^{(0)} \in \mathbb{R}^{d \times M}$ be the initialization of the pooled graph representation and C_1, C_2 be the $M \times N$ cost matrices that capture the pairwise squared Euclidean distances from $X^{(0)}$ to Y_1 and to Y_2, respectively. From Eq.(4.18) we obtain:

$$C_2 = C_1 \mathscr{P}_f. \tag{4.19}$$

OT coupling

Because of the entropic regularization, the problem in Eq. (2.11) is a strictly convex minimization problem. Given uniform weight distributions, $a = \frac{1}{M}\mathbb{1}_M$ and $b = \frac{1}{N}\mathbb{1}_N$, the only variable controlling the unique solution P^* is the cost for the transportation. Therefore, due

to the relationship between the costs C_1, C_2 in Eq.(4.19), it holds that:

$$P_2^* = P_1^* \mathscr{P}_f. \tag{4.20}$$

Gradient

Due to Eq. (4.19), (4.20), and the expression for the gradient of $L_C^\epsilon(a,b)$ with respect to X in Eq. (4.17), it follows that the gradient $\nabla_X L^\epsilon_{C(X,Y)}(a,b)$ will be the same for both the permuted and the non-permuted case.

Update Step

As a result, if the initialization $X^{(0)}$ is common in both cases, then $X_1^{(1)} = X_2^{(1)}$. Similarly, we can show that this is true for all iterations $0 < l < L$ of the flow. Therefore, it holds that $X_1^{(L)} = X_2^{(L)} \Leftrightarrow \text{FlowPool}(Y_1) = \text{FlowPool}(Y_2)$. The assumption that $X^{(0)}$ is common for both cases is a reasonable one. In fact, as we demonstrate in Section 4.8, it is necessary for $X^{(0)}$ to be shared among all graph representations in order for our proposed pooling method to perform well. In Section 4.9, we discuss how one can go a step further by parametrizing $X^{(0)}$ and learning it from the data.

4.7 Integrating FlowPool in Graph Neural Network Architectures

In order to integrate FlowPool in end-to-end deep learning architectures for graphs an element that needs to be considered is the differentiation of the output of FlowPool with respect to its input. This differentiation is key in the general setting, where the input representation Y can be the output of one or more layers with trainable weights and we need to backpropagate through FlowPool in order to learn these weights. In order to backpropagate through FlowPool we need to compute the Jacobian of the output $X^{(L)}$ with respect to the input Y. In this section we outline how this computation is handled. We depict a block-scheme of the computations in FlowPool in Fig. (4.3).

The `OTT` toolbox used in this work is based on the automatic differentiation framework JAX [98]. In the forward pass of FlowPool the gradient $\nabla_X L^\epsilon_{C(X,Y)}(a,b)$, computed using the implicit differentiation scheme described above, is returned by calling JAX's function `jax.grad` on the entropy regularized cost $L^\epsilon_{C(X,Y)}(a,b)$. In Fig. (4.3) we show how the pooled representation after the first step of the flow, $X^{(1)}$, is computed using the gradient $\nabla_X L^\epsilon_{C(X,Y)}(a,b)$ evaluated at $X^{(0)}$ and Y. At the backward pass, we need to differentiate again the function $\nabla_X L^\epsilon_{C(X,Y)}(a,b)$. We note that the enveloppe theorem does not hold for higher order derivatives and, therefore, the second term in Eq. (4.15) will be re-differentiated during

the backpropagation. The re-derivation of the gradient function obtained with `jax.grad` is possible using JAX's ability to perform automatic differentiation of the linear system in Eq. (4.3). Specifically, this is accomplished with JAX's function `jax.lax.custom_linear_solve` that wraps all linear solvers in JAX [92] and allows to compute higher order derivatives, implicitly again.

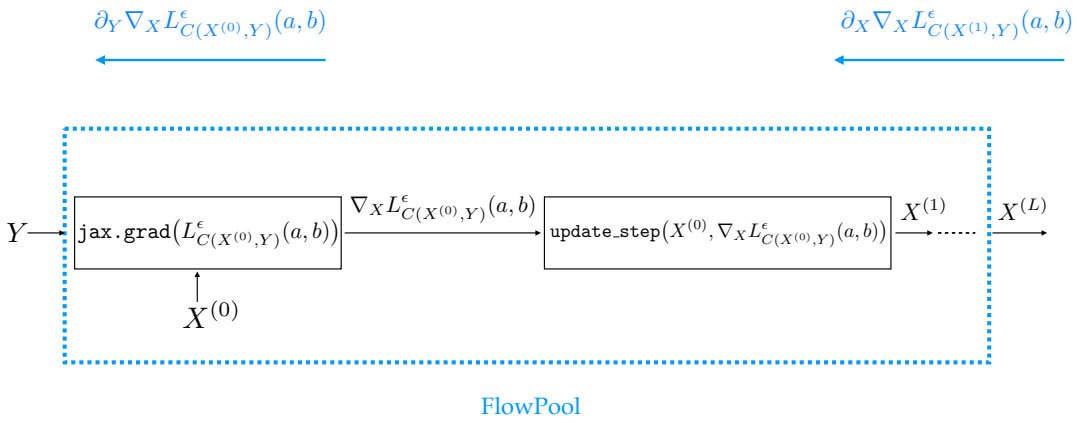


Figure 4.3 – Implementation of FlowPool using JAX. The gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ needed for the gradient flow is obtained with JAX's function `jax.grad`. The pooled graph representation is updated according to this gradient. During the backpropagation the gradient function obtained with `jax.grad` is re-derived in order to obtain the Jacobians $\partial_Y \nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ and $\partial_X \nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$.

At the backpropagation of the first step of the flow, the Jacobian needed is $\partial_Y \nabla_X L^{\epsilon}_{C(X^{(0)},Y)}(a,b)$, which captures how the gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ is affected by changes in the input graph representation Y, for the initialization of the pooled representation $X^{(0)}$. In the next steps of the flow, Jacobians of the form $\partial_X \nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ are also needed. We note that $\partial_X \nabla_X L^{\epsilon}_{C(X,Y)}(a,b) = \nabla^2_X L^{\epsilon}_{C(X,Y)}(a,b)$, where ∇^2 stands for the Hessian matrix with second-order derivatives. The computation of the Jacobian of $X^{(1)}$ with respect to $\nabla_X L^{\epsilon}_{C(X^{(0)},Y)}(a,b)$ is straightforward. For instance, in the case of a gradient update step of the form $X^{(1)} = X^{(0)} - \tau \nabla_X L^{\epsilon}_{C(X^{(0)},Y)}(a,b)$, it will be equal to $-\tau I$. Therefore, using JAX's automatic differentiation we can compute the Jacobian of $X^{(L)}$ with respect to Y, and therefore backpropagate through FlowPool in order to train the layers that provide the graph representation Y[1].

[1]We would like to thank Marco Cuturi for his kind interaction through the Github issues of the `OTT` toolbox and for the modifications in the source code that enabled the computation of second-order derivatives.

4.8 Experiments

4.8.1 Experimental Analysis

In this Section we analyse the behaviour of FlowPool and the dependence on its parameters. For this analysis we consider the simplified setting where FlowPool is not integrated in a deep network. Further, we use the squared Euclidean distance cost in order to visualize the representations and build intuitions with regards to the way that FlowPool operates.

Experimental Settings

We consider two chemical compound datasets BZR and COX2 [99]. The chemical compounds are represented by graphs, where nodes correspond to atoms and edges represent chemical bonds. Each node is described by a 3-dimensional feature vector. The chemical compounds considered are benzodiazepine receptor ligands in BZR and cyclooxygenase-2 inhibitors in COX2. Further, the graphs are labeled as active or inactive against some type of cell. Therefore, the datasets can be used for binary graph classification. We provide the statistics of the datasets in Table 4.1.

Table 4.1 – Statistics of Datasets

Dataset	# graphs	# classes	avg. node #	avg. edge #	# node features
BZR	405	2	35.75	38.36	3
COX2	467	2	41.22	43.45	3

In our experimental analysis, the representation Y of a graph corresponds to the features of its nodes, as provided by the dataset. As a result, a graph of N nodes corresponds to a representation $Y \in \mathbb{R}^{3\times N}$.

The setup considered for our analysis is shown in Fig. (4.4). We consider the input to FlowPool to be a graph representation Y. FlowPool is used to pool Y to the fixed-size representation $X^{(L)}$, which is input to a logistic regression classifier. We train the classifier by minimizing the cross-entropy loss with an l_2 regularization on the weights, in order to avoid over-fitting. The weights w and intercept c of the classifier are the only learnable parameters. We denote the sigmoid function as:

$$s(\theta) = \frac{e^{\theta}}{e^{\theta}+1}. \tag{4.21}$$

The considered datasets are not split into pre-defined training and test sets. Therefore, we evaluate the classification task over 10 different partitions of the data using 10 folds, as in a 10-fold cross validation scheme. For each partition, we use the 9 folds as our training data

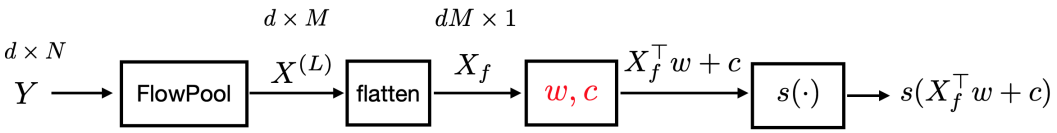


Figure 4.4 – Block scheme of graph classification with FlowPool. FlowPool is used to pool a provided graph representation Y to a graph representation of fixed size $X^{(L)}$. The representation $X^{(L)}$ is flattened to obtain the representation X_f. The learnable parameters are the weights w and the intercept c of the logistic regression classifier, which are annotated with red. The features in X_f are linearly combined and passed through the sigmoid function, in order to obtain the probability that the graph representation Y belongs to the positive class.

and the remaining fold for evaluation, and provide the mean and standard deviation of the classification accuracy for the 10 partitions. Further, we notice that the datasets BZR and COX2 are characterized by a class imbalance, as shown in Table 4.2. This is a common issue in biological datasets, such as the ones considered here. In order to properly take into account this imbalance, we consider stratified splits and we compute the balanced classification accuracy [100]. In the binary case, the balanced accuracy is equal to the arithmetic mean of sensitivity (true positive rate) and specificity (true negative rate):

$$\text{balanced accuracy} = \frac{1}{2}\Big(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\Big), \tag{4.22}$$

where TP, FP, TN, FN stand for true positives, false positives, true negatives and false negatives, respectively.

Table 4.2 – Class Imbalance

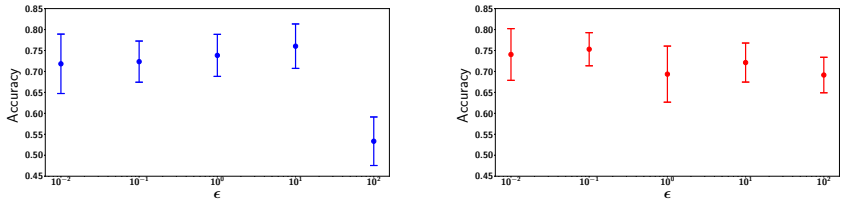
Dataset	% positive class	% negative class
BZR	21.23	78.77
COX2	21.84	78.16

In what follows, we study the impact of the entropy regularization parameter and the removal of the bias of the entropic regularization. Further, we discuss the effect of the initialization of the pooled graph representations. Finally, we evaluate the performance of FlowPool on a binary classification task by comparing to other graph pooling methods.

Impact of Entropy Regularization

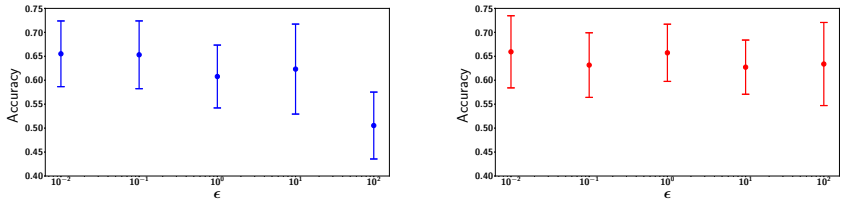
First, we study the impact of the entropy regularization parameter on the pooled graph representation. In order to do so, we consider values of the regularization parameter in

$\epsilon = [0.01, 0.1, 1, 10, 100]$. Using the classification set-up of Fig. (4.4), with the same stratified splits for all the evaluations and for $M = 10$, we compute the mean and the standard deviation of the balanced classification accuracies over the 10 partitions of the data. We set the number of iterations of the gradient flow to $L = 200$ and the step size to $\tau = 0.2$. The results are shown in Fig. (4.5a) and Fig. (4.6a) for the BZR and COX2 datasets respectively. It can be seen that the value of ϵ affects the classification accuracy obtained. Further, we notice that for $\epsilon = 100$ there is a significant drop in the performance for both datasets. In order to explain why this is the case, we demonstrate in Fig. (4.7), (4.8) the representations obtained with FlowPool for different values of ϵ for the BZR and COX2 datasets, respectively. We show with blue the initial representation Y input to FlowPool and with red its pooled counterpart after L iterations of the flow. It can be seen that, as the values of the entropic regularization become larger, the pooled embeddings tend to"collapse". In the extreme case of $\epsilon = 100$, the pooled representation yields the mean value of Y, since we use the squared Euclidean distance cost, and our method becomes equivalent to mean pooling. Further, we notice that for COX2 there is a clear trend of a dampening in its performance for larger values of ϵ, while for BZR there is a small increase for $\epsilon = 10$. We believe that this indicates that a smaller pooling dimension M may be the optimal for this dataset.



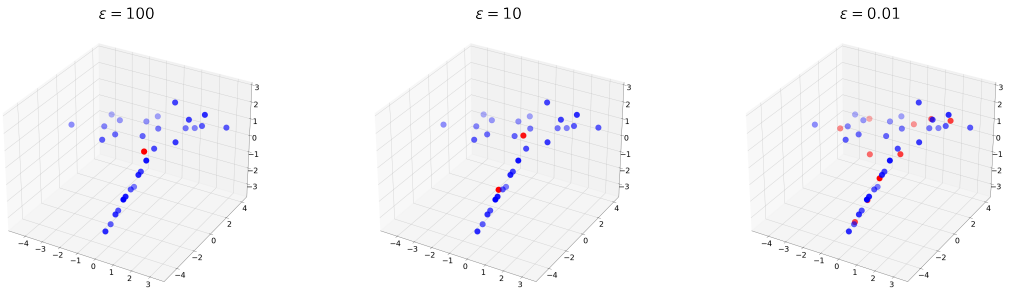
(a) $L_C^\epsilon(a,b)$ (b) $S_C^\epsilon(a,b)$

Figure 4.5 – Impact of the entropy regularization parameter ϵ on the classification accuracy for the BZR dataset using FlowPool for the minimization of (a) the entropy regularized Wasserstein distance $L_C^\epsilon(a,b)$ and (b) the Sinkhorn divergence $S_C^\epsilon(a,b)$.



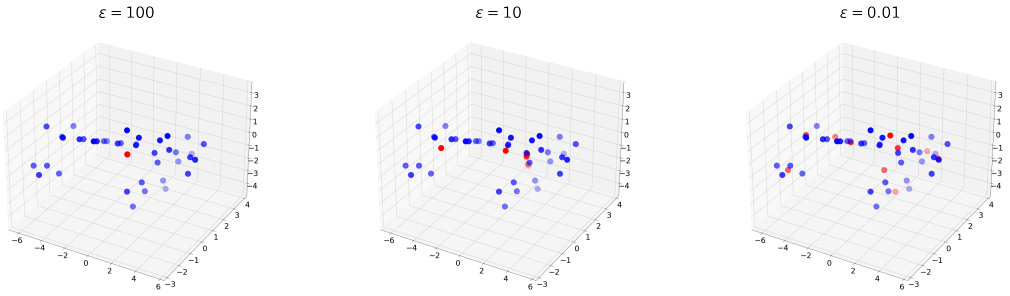
(a) $L_C^\epsilon(a,b)$ (b) $S_C^\epsilon(a,b)$

Figure 4.6 – Impact of the entropy regularization parameter ϵ on the classification accuracy for the COX2 dataset using FlowPool for the minimization of (a) the entropy regularized Wasserstein distance $L_C^\epsilon(a,b)$ and (b) the Sinkhorn divergence $S_C^\epsilon(a,b)$.



(a) (b) (c)

Figure 4.7 – Pooled graph representations for different values of ϵ for a representation from BZR. With blue we show the initial representation $Y \in \mathbb{R}^{3\times 30}$ and with red the pooled graph representation $X^{(L)} \in \mathbb{R}^{3\times 10}$. Larger values of ϵ result to degenerate pooled representations $X^{(L)}$ with smaller support than Y.



(a) (b) (c)

Figure 4.8 – Pooled graph representations for different values of ϵ for a representation from COX2. With blue we show the initial representation $Y \in \mathbb{R}^{3\times 39}$ and with red the pooled graph representation $X^{(L)} \in \mathbb{R}^{3\times 10}$. Larger values of ϵ result to degenerate pooled representations $X^{(L)}$ with smaller support than Y.

Interestingly the impact of the parameter ϵ has a very intuitive interpretation. We remind that:

$$L_C^{\epsilon}(a,b) = \min_{P\in U(a,b)} \langle C, P\rangle - \epsilon H(P). \tag{4.23}$$

Equivalently, as discussed in [101], we can write:

$$L_C^{\epsilon}(a,b) = \min_{P\in U(a,b)} \langle C, P\rangle + \epsilon \,\mathrm{KL}(P|ab^{\top}). \tag{4.24}$$

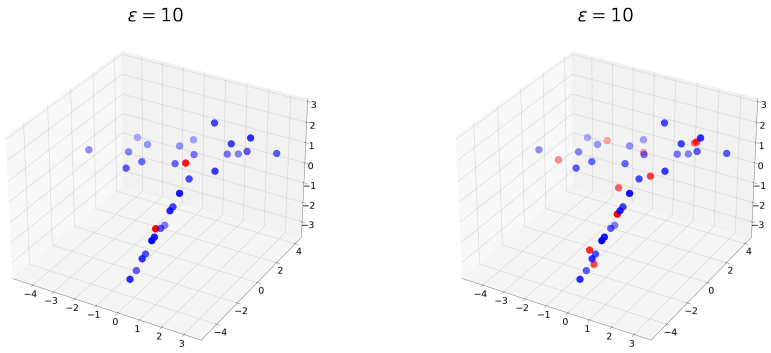
We can observe that the quantity $\mathrm{KL}(P|ab^\top)$ is equal to the mutual information of the probability measures ν and μ that correspond to the input graph representation and its pooled counterpart respectively (Section 4.3). The mutual information is minimal, and equal to zero, when the measures μ, ν are independent and the probabilistic coupling corresponds to the independence matrix $P = ab^\top$. As can be seen from Eq. (4.23), the optimal coupling will correspond to $ab^\top$ when $\epsilon \to \infty$. As a result, in order to maximize the mutual information between μ and ν, while enjoying the properties of well defined gradients provided by the entropic regularization, one should chose small values for ϵ. This can be done without getting numerical issues using the log-stabilized iterations in Eq. (2.23).

The collapse of the pooled graph representation, as demonstrated in Fig. (4.7), (4.8), is due to what is commonly referred to as the entropic bias. The entropy-regularized optimal transport loss between a measure $\nu = \sum_{j=1}^{N} b_j \delta_{y_j}$ and itself is equal to $L^{\epsilon}_{C(Y,Y)}(b,b)$, where $C(Y,Y)$ denotes the $N \times N$ cost matrix with the pairwise distances between the points $\{y_j\}_{j=1}^{N}$. For $\epsilon > 0$, $L^{\epsilon}_{C(Y,Y)}(b,b) \neq 0$ and, as a result, the minimization of $L^{\epsilon}_{C(X,Y)}(a,b)$ with respect to the positions of the measure μ leads to a biased solution, with the measure μ having a smaller support than that of the target measure ν. In [101], it is shown that a principled way to remedy that is to minimize instead the Sinkhorn divergence:

$$S^{\epsilon}_{C(X,Y)}(a,b) = L^{\epsilon}_{C(X,Y)}(a,b) - \frac{1}{2} L^{\epsilon}_{C(X,X)}(a,a) - \frac{1}{2} L^{\epsilon}_{C(Y,Y)}(b,b). \tag{4.25}$$

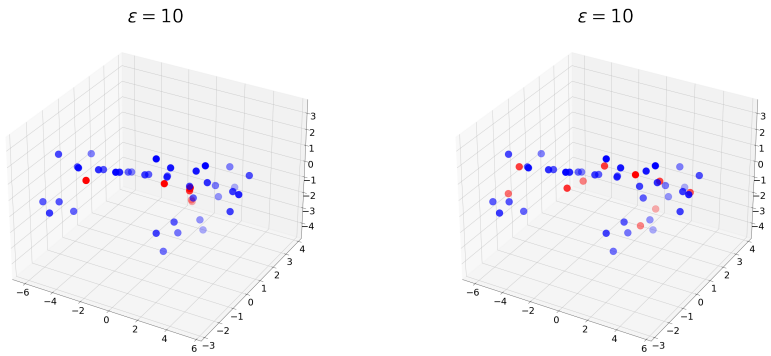
We consider again the experiment described above for values of the regularization parameter in $\epsilon = [0.01, 0.1, 1, 10, 100]$, while substituting the computation of $L^{\epsilon}_{C(X,Y)}(a,b)$ with the Sinkhorn divergence $S^{\epsilon}_{C(X,Y)}(a,b)$. The values of τ and L are kept the same. We show in Fig. (4.5b), (4.6b) the classification accuracies obtained for the BZR and COX2 datasets, respectively. It can be seen that, since the entropic bias has been removed, the impact of the value of ϵ on the classification accuracy is practically eliminated. We show in Fig. (4.9), (4.10), examples of the representations obtained when the energy minimized is that of the entropy regularized Wasserstein distance and that of the Sinkhorn divergence. It can be seen that for the case of the Sinkhorn divergence, the pooled representation is not degenerate, even for the large value of $\epsilon = 10$. Therefore, the minimization of the Sinkhorn divergence constitutes FlowPool less sensitive to the hyperparameter ϵ and improves the quality of the pooled representation. Further, the Sinkhorn divergence offers an advantage in terms of computational complexity. This is due to the fact that larger values of ϵ can be considered without the "collapse" of the pooled embedding. As less iterations are needed for Sinkhorn to converge with larger values of ϵ, it follows that the Sinkhorn divergence is computationally more advantageous. A discussed in [101], the autocorrelation terms in Eq. (4.25) converge in a very small number of iterations and, therefore, their computational overhead is negligible. We note that in the case of FlowPool, only the autocorrelation term $L^{\epsilon}_{C(X,X)}(a,a)$ provides non-zero gradients with respect to X.

To sum up, the Sinkhorn divergence should be preferred over the entropy-regularized Wasserstein distance since it constitutes FlowPool less sensitive to the hypeparameter ϵ and less computationally expensive. The optimal values of ϵ and M can be cross-validated based on the data under consideration and the task at hand.



(a) $L^{\epsilon}_{C(X,Y)}(a,b)$ (b) $S^{\epsilon}_{C(X,Y)}(a,b)$

Figure 4.9 – Comparison of the result of FlowPool on the feature representation of a graph from the BZR dataset for (a) the entropy regularized Wasserstein distance $L^{\epsilon}_{C(X,Y)}(a,b)$ and (b) the Sinkhorn divergence $S^{\epsilon}_{C(X,Y)}(a,b)$ for $\epsilon = 10, \tau = 0.2, L = 200$. It can be seen that the Sinkhorn divergence yields an unbiased solution even for this large value of $\epsilon = 10$.



(a) $L^{\epsilon}_{C(X,Y)}(a,b)$ (b) $S^{\epsilon}_{C(X,Y)}(a,b)$

Figure 4.10 – Comparison of the result of FlowPool on the feature representation of a graph from the COX2 dataset for (a) the entropy regularized Wasserstein distance $L^{\epsilon}_{C(X,Y)}(a,b)$ and (b) the Sinkhorn divergence $S^{\epsilon}_{C(X,Y)}(a,b)$ for $\epsilon = 10, \tau = 0.2, L = 200$. It can be seen that the Sinkhorn divergence yields an unbiased solution even for this large value of $\epsilon = 10$.

Initialization of the Pooled Representation

The initialization $X^{(0)}$ impacts directly the pooled representation output by FlowPool. This is due to the fact that the optimization problem in Eq. (4.2) is non-convex with respect to X. Further, we notice that in order for FlowPool to yield meaningful representations, the initialization $X^{(0)}$ must be shared among all input graph representations Y. In order to demonstrate this, we consider the experimental setting of Section 4.8.1 with $M = 10$. We set $\tau = 0.2$, $L = 200$, $\epsilon = 0.1$ and use the Sinkhorn divergence. We compute the classification accuracy for BZR and COX2 for the case where the initialization is fixed, and shared among all representations, and for the case where the initialization varies and is different for each input representation. The obtained accuracies using the same stratified folds are shown in Table 4.3. Three different initialization seeds are used in order to smooth the effect of unfavorable random initialization. It can be seen that the accuracies obtained when the initialization is shared among all inputs are significantly higher than the ones obtained for the case where the initialization is different for each input.

Table 4.3 – Impact of Fixed Initialization.

Dataset	FlowPool (f. i.)	FlowPool
BZR	**72.51 ± 5.57**	54.82 ± 6.49
COX2	**63.24 ± 5.18**	50.96 ± 7.24

The reason why this is the case is the following. Let us consider a representation Y_1, with pairwise distances from $X^{(0)}$, captured by the cost C_1. If a representation Y_2, that is close to Y_1, is considered, the cost matrix C_2 will have similar values to those of C_1. As a result, the optimal coupling P_1^* will be close to P_2^*. Given the same initialization $X^{(0)}$, due to Eq. (4.17), the gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ evaluated at $(X^{(0)}, Y_1)$ will have similar values to that evaluated at $(X^{(0)}, Y_2)$ and consequently the pooled representations $X_1^{(1)}$ and $X_2^{(1)}$ will also be close. We can reason in a similar way for the next steps of the flow. Therefore, by using the same initialization for the source of the two transportation problems, we manage to obtain a sense of the similarity of the input representations Y_1, Y_2.

Graph Classification

Having analysed the impact of the parameters of FlowPool on its performance, we now evaluate its efficiency in pooling graph representations compared to other methods. In order to evaluate the effectiveness of FlowPool, we consider the classification set-up of Fig. (4.4) and compare to other methods by substituting the FlowPool block with the following node clustering methods:

- K-means++ clustering [102]: The K-Means algorithm clusters the N node representations in $Y \in \mathbb{R}^{d\times N}$ by separating them into groups of equal variance. The centroids are initialized with the K-means++ initialization [103].

- K-means clustering with fixed random initialization: The K-means algorithm, where the centroids of the clusters are initialized randomly, but are shared across all input representations.
- Spectral Clustering (SC)[104]: SC performs a low-dimensional embedding of the Laplacian matrix of a graph, followed by clustering of the components of the eigenvectors in the low dimensional space.
- DiffPool [78]: DiffPool learns assignment matrices from the data by parametrizing them as $S = \text{softmax}(\text{GCN}(A, F))$, where A and F correspond to the adjacency matrix and the features of a graph, respectively. GCN corresponds to the graph convolutional filter [67] defined as $\text{GCN}(A, F) = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}F\Theta$, where $\Theta \in \mathbb{R}^{d\times M}$ are the trainable weights, $\tilde{A} = A + I_N$ and $\tilde{D}$ is the diagonal matrix of node degrees of $\tilde{A}$.
- MinCutPool [83]: MinCutPool parametrizes the assignment matrices as $S = \text{softmax}(F\Theta)$, where $\Theta \in \mathbb{R}^{d\times M}$ are the weights that are learned from the data. A regularizer, which approximates the relaxed formulation of the minCUT problem, affects the learned values of the trainable parameters Θ.

In the set-up we consider, we do not learn $X^{(0)}$, but initialize it randomly. Therefore, in order to compare to node clustering pooling methods, we consider fixed and random weights for their parameters as well. In order to mitigate the effect of possible favourable or unfavourable initializations on test performances, we consider three different random seeds for the initialization. For K-means++ the random initialization will affect the selection of the cluster centers from the datapoints. In the case of K-means with fixed random initialization, the seeds will affect the initialization of the centroids that is shared among all input representations. In the case of Spectral Clustering, the random initialization pertains to the initialization of the clusters in the low-dimensional space. For DiffPool, MinCutPool and FlowPool the random initialization affects the parameters of the GCN layer, the fully connected layer and the initialization of the source $X^{(0)}$, respectively. For FlowPool we set $\tau = 0.2$, $L = 200$, $\epsilon = 0.1$ and use the Sinkhorn divergence. All methods are compared over the same stratified splits and the same random seeds. The classification accuracies are averaged over the ten partitions and the three different random initializations. The mean and the standard deviation of the obtained accuracies for all methods are shown in Table (4.4). It can be seen that, for both datasets, FlowPool yields the highest classification accuracy. For the BZR dataset, DiffPool and MinCutPool achieve the second best performance and their accuracies are comparable. We notice that Spectral Clustering does not perform well in pooling the node representations for the BZR dataset. For the COX2 dataset, the second best performance is achieved by K-means with fixed initialization, followed by SC. The reason why K-means with fixed initialization performs relatively well for both datasets is due to the fact that there is a close relationship between K-means and the minimization of the unregularized Wasserstein distance (as in Eq. (2.10)). This relationship is studied in [105], [106]. Finally, we

observe that K-means++ performs poorly for both datasets. The reason for this stems from the fact that the K-means++ initialization is an algorithm that selects in a systematic way K points of the data as the centroids for the clustering. Therefore, the initialization for K-means in that case is not common for all input representations and, as a result, performance is hindered.

Table 4.4 – Graph Classification Accuracy.

Dataset	K-means++	K-means (f.i.)	SC	DiffPool	MinCutPool	FlowPool
BZR	50.80 ± 6.97	68.87 ± 7.41	59.24 ± 7.98	71.04 ± 4.85	70.64 ± 5.05	**72.51 ± 5.57**
COX2	54.09 ± 8.38	62.73 ± 7.62	61.09 ± 7.84	60.77± 8.59	60.56 ± 8.14	**63.24 ± 5.18**

In the comparison carried out here, the only methods that use the graph structure are SC and DiffPool. MinCutPool, in the scenario of random weights considered, does not use structural information. This is because the minCUT regularizer can only impact the parameters Θ during training. FlowPool also does not take into account the graph structure as the cost C considered here employs only the similarity of the features, as captured by the squared Euclidean distance.

4.8.2 Preliminary Classification Results of FlowPool in a GNN

We now use FlowPool in order to perform global pooling in a GNN architecture. For this experiment we use the PROTEINS [107], [108] dataset. Proteins are represented as graphs, where nodes correspond to secondary structure elements (helix, sheet, turn) and two nodes are connected by an edge if they are neighbors along the amino-acid sequence or one of three nearest neighbors in space. The dataset consists of 1113 graphs with average node number $\bar{N} = 39.06$ and average edge number $\bar{E} = 72.82$. Further, each node is labeled according to its type using a one-hot encoded vector. We perform a binary classification task where we predict whether proteins are enzymes or not.

In order to do so, we consider a GNN composed of a graph convolutional layer [67], a global pooling layer and a fully connected layer followed by a sigmoid activation. The network is trained by minimizing the binary cross-entropy loss[2]. We use the same assessment method with 10-fold CV and the same stratified splits as those proposed in the paper for fair comparison of GNNs in the task of graph classification [112]. With this experiment we are solely interested in assessing the possibility of the integration of FlowPool in end-to-end deep learning architectures. Therefore, no model selection is performed in order to optimally tune hyperparameters. We keep the GNN architecture fixed and compare the classification accuracy obtained when using as the global pooling layer DiffPool, MinCutPool and FlowPool with random weights. All models are trained with Adam [71] with a learning rate of $\mu = 0.0001$ and batch size = 8. The embedding dimension is set to $d = 3$ in order for us to visualize the

[2]Software: For the implementation of the GCN layer we use J-Raph [109]. For the gradient processing and optimization we use Optax [110]. The neural network library used is Haiku [111].

representations and ensure that our method functions as expected when integrated in a GNN. The size of the pooled representation is set to $M = 20$. For FlowPool we set $\epsilon = 0.2$, $\tau = 0.1$ and $L = 2000$ and use the Sinkhorn divergence in order to remove the entropic bias. The pooled graph's representation $X^{(0)}$ is set by randomly selecting M samples from the normal distribution $\mathcal{N}(0_d, I_d)$. We train over 1000 epochs and perform early-stopping [113] with a patience of 50 epochs in order to avoid overfitting. The criterion for early-stopping is based on the validation loss. We report the mean and the standard deviation of the classification accuracy over the 10 folds in Table 4.5. It can be seen that our proposed pooling method performs on par with MinCutPool and offers a small advantage in performance compared to DiffPool. We note that the performance of all methods can be improved by optimally selecting their hyperparameters. We expect that with optimal tuning, FlowPool will perform significantly better than other methods, as in Section 4.8.1. With this preliminary experiment, we have, however, performed a sanity check of the possibility of integrating FlowPool in end-to-end deep learning architectures for graphs.

Table 4.5 – Classification Accuracy for PROTEINS.

Dataset	DiffPool	MinCutPool	FlowPool
PROTEINS	69.27 ± 5.77	71.26 ± 3.87	**71.45 ± 4.22**

In Fig. (4.11) we show an example of a representation and its pooled version, as obtained with FlowPool, in this experiment. It can be seen that our pooling method behaves as expected, but that extra tuning of its parameters is needed in order to obtain a pooled representation that optimally preserves the statistics. This could be due to the fact that the range of values of the representations Y are smaller than those of Section 4.8.1, as can be seen by noticing the difference in ranges in Fig. (4.7) and Fig. (4.11). Also, due to the smallest range of values of the representations in that case, it may make sense to adapt the value of ϵ in order to obtain discriminative Gibbs kernels $K = e^{-\frac{C}{\epsilon}}$.

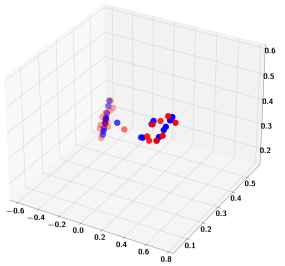


Figure 4.11 – Pooling of a GCN representation with FlowPool. With blue we show the initial representation Y and with red the pooled graph representation $X^{(L)}$.

Further, we highlight that the performance of a GNN network with FlowPool depends on the solution $\partial[f^*(X), g^*(X)]$ of the linear system that occurs during the implicit differentiation. In the forward pass, the solution of this linear system provides the gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$ via Eq. (4.4). In the backward pass, it affects the propagation of the derivatives through the Jacobians $\partial_Y \nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$, $\partial_X \nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$, which also depend on $\partial[f^*(X), g^*(X)]$ through Eq. (4.4). Therefore, in order to guarantee that both the forward and the backward pass behave as expected, we must ensure that the possible rank deficiency of the matrix in Eq. (4.8) is properly accounted for by selecting appropriate values for the regularizers that enforce the stability of the linear system. Finally, we note that during the backward pass and, therefore, the differentiation of $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$, the second term in Eq. (4.15) affects the Jacobians $\partial_Y \nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$, $\partial_X \nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$. The Jacobian $\partial_C P^*(C)$ that captures how small changes in the cost C affect the optimal solution P^*, can be numerically unstable. This can be more prevalent for small values of ϵ. In the extreme case where $\epsilon \to 0$, the solution $L^{\epsilon}_C(a,b)$ tends to the solution $L_C(a,b)$ of the unregularized optimal transport problem of Eq. (2.7). As discussed in Section 2.3, the unregularized OT problem is a linear program and, therefore, attains its optimal solution on one of the vertices of the polytope $U(a,b)$ defined by the mass-preservation constraints. As a result, for small values of ϵ, small changes in the cost C can cause a significant change in P^*. This instability of $\partial_C P^*(C)$ does not affect the values of the gradient $\nabla_X L^{\epsilon}_{C(X,Y)}(a,b)$, needed in the forward pass, because of the envelope theorem, as discussed in Section 4.6.

By working on the numerical issues discussed, we expect the performance of FlowPool when integrated in GNNs to become significantly better than that of competing methods. Finally, we underline that the squared Euclidean distance was used as a cost matrix in these experiments mostly in order to provide intuitive visualizations and comprehension of the way that FlowPool operates. The investigation of different costs is possible due to the automatic differentiation framework discussed in Sections 4.5, 4.7.

4.9 Parametrization of FlowPool - Learning the Ground Cost

The pooled representations returned by FlowPool can become more relevant to the specific dataset, used to train the GNN architecture, by considering a parametrization of the ground cost $C = C(X,Y)$ used for the mass transportation. Two types of parametrization are possible. The first is to consider a known cost function $C(X,Y)$, such as the squared Euclidean cost used in the previous section, and learn the initialization $X^{(0)} \in \mathbb{R}^{d \times M}$ from the data. The second option is to consider a parametrization of the function $C(X,Y)$ in order to take into account the structures of the considered graphs. This corresponds to the problem of ground metric learning and has been studied in a line of works, such as [114], [115], [116], [117], among others. We believe that learning an appropriate cost from the data could offer a promising direction for future work. Our implementation of FlowPool with automatic differentiation

ensures that the relevant derivatives can be computed for any cost C that captures the pairwise relationship between the input representation and its pooled counterpart.

4.10 Conclusion

In this chapter we proposed FlowPool, a framework for pooling graph representations, while optimally preserving their statistical properties. Our proposed method is framed as a Wasserstein gradient flow in the graph representation space and admits an intuitive parametrization. We proposed a versatile implementation of our method, based on automatic differentiation, that can take into account the geometry of the representation space through any optimal transport cost. We performed an experimental analysis of FlowPool on simplified settings and showed promising results on graph classification. Finally, we demonstrated that our method can be integrated in end-to-end deep learning architectures for graphs and provided directions for future work.

5 Conclusion and Future Work

In this thesis we introduced new methods to take into account geometrical information in order to enhance the performance of end-to-end deep architectures for graph representation learning. We used probability measures in order to describe graph data and leveraged the theory of Optimal Transport in order to define geometrically meaningful distances between them. We proposed effective methods for both transductive [118] and inductive graph representation learning settings. For the transductive setting we have access to a given graph during the representation learning process. Therefore, the geometry of interest in that case is fixed and discrete, as captured by the pairwise relationships between the nodes. As a result, for that setting, we propose a Eulerian discretization and correspond the graph data to probability measures of fixed support or, equivalently, to histograms. For the inductive setting, where the representation learning algorithm is expected to generalize to unkown graph instances, the geometry of interest is continuous. Thus, in this setting, we use a Lagrangian discretization and correspond graph data to probability measures of free support. The achievements of this thesis are summarized below.

First, we consider the transductive setting where the data correspond to a given graph structure. We propose a Wasserstein barycentric method that can provide non-linear and geometry aware interpolations of the histograms that describe the data. We introduce an efficient implementation of our method in terms of time computational complexity and explain how it can be differentiated in an automatic way by unrolling the Sinkhorn iterations of the Wasserstein barycenter computation. We then proceed to integrate the proposed method in an autoencoder architecture with the goal of learning node embeddings that are directly interpretable and robust to perturbations of the graph structure. In order to achieve this goal, we cast our representation learning problem to that of learning simultaneously i) a low-dimensional space and ii) coordinates for the nodes in that low-dimensional space. The low-dimensional space contributes to interpretability as the representations that define it capture the most relevant structural information in the graph. Further, it is conducive to stability, as it can be used to embed perturbed graphs and, therefore, offers the possibility to

register the embeddings of the clean graph and its perturbed versions. The coordinates of the nodes are directly interpretable as their values reveal the proximity to each representation that defines the low-dimensional space. We validate experimentally our proposed method and show that it is stable to perturbations of the graph structure and achieves comparable or superior results compared to state-of-the-art unsupervised graph representation learning methods on the task of node classification. We believe that an interesting direction for extending this work would be to study how the barycentric layer can be incorporated in other deep learning architectures for graphs, apart from our proposed autoencoder. We believe that the exploration of different costs, apart from the diffusion distance, could be relevant in that case. Also, the creation of feature extraction layers that could apply more localized Wasserstein barycenter computations could be promising.

Second, we focus on the inductive setting where the data is composed of a set of graphs. In this setting, graphs can naturally be of varying size. Therefore, in order to enable end-to-end graph representation learning, a global pooling operation is needed. We introduce a pooling method that optimally preserves the statistics by minimizing the Wasserstein distance between a graph representation and its pooled version. In order to do so, we perform a Wasserstein gradient flow with respect to the positions of a source of fixed size, which corresponds to the pooled representation. Further, we propose a versatile and efficient implementation for our pooling method. Specifically, we compute the gradient of the energy of the Wasserstein distance with respect to the positions, using recently introduced automatic differentiation schemes that employ the implicit function theorem. Due to the automatic differentiation used, our proposed method is versatile and can be used with any cost for the mass transportation. Further, we demonstrate how our method is directly amenable to backpropagation and can therefore be incorporated in end-to-end deep networks. This is possible by computing higher order derivatives of the Wasserstein distance during the reverse-mode automatic differentiation. We provide an experimental analysis of our pooling method and build intuitions with respect to its functionality. Further, we evaluate it in graph classification tasks and show that our proposed pooling method provides promising results. Interesting future work would be to parametrize the ground cost used for the mass transportation and learn it from the data at hand. We believe that this could provide also useful intuitions with regards to what feature information is relevant for the representation learning process.

To conclude, this thesis offers new ways of infusing geometrical information in end-to-end deep architectures for graphs by employing elements from the Optimal Transport theory. It provides a mathematical analysis, builds intuitions and demonstrates important empirical results in relevant applications of graph representation learning.

Bibliography

[1] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning (ICML*. PMLR, 2017, pp. 1263–1272.

[2] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[3] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2018.

[4] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Syx72jC9tm

[5] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.

[6] R. R. Coifman and S. Lafon, "Diffusion maps," *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 5–30, 2006.

[7] E. Simou, D. Thanou, and P. Frossard, "node2coords: Graph representation learning with wasserstein barycenters," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 17–29, 2020.

[8] G. Peyré, M. Cuturi *et al.*, "Computational optimal transport," *Foundations and Trends® in Machine Learning*, vol. 11, no. 5-6, pp. 355–607, 2019.

[9] G. Monge, "Mémoire sur la théorie des déblais et des remblais," *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

[10] L. Kantorovich, "On translation of mass," *Doklady Akademii nauk SSSR*, vol. 37, pp. 227–229, 1942.

[11] D. Goldfarb and J. K. Reid, "A practicable steepest-edge simplex algorithm," *Mathematical Programming*, vol. 12, no. 1, pp. 361–371, 1977.

[12] C. Villani, *Optimal transport: old and new*. Springer Science & Business Media, 2008, vol. 338.

[13] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 2292–2300.

[14] P. A. Knight, "The sinkhorn–knopp algorithm: convergence and applications," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 261–275, 2008.

[15] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[16] B. Schmitzer, "Stabilized sparse scaling algorithms for entropy regularized transport problems," *SIAM Journal on Scientific Computing*, vol. 41, no. 3, pp. A1443–A1481, 2019.

[17] L. Chizat, G. Peyré, B. Schmitzer, and F.-X. Vialard, "Scaling algorithms for unbalanced optimal transport problems," *Mathematics of Computation*, vol. 87, no. 314, pp. 2563–2609, 2018.

[18] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[19] M. Agueh and G. Carlier, "Barycenters in the wasserstein space," *SIAM Journal on Mathematical Analysis*, vol. 43, no. 2, pp. 904–924, 2011.

[20] H. Janati, M. Cuturi, and A. Gramfort, "Wasserstein regularization for sparse multi-task regression," in *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019, pp. 1407–1416.

[21] J.-D. Benamou, G. Carlier, M. Cuturi, L. Nenna, and G. Peyré, "Iterative bregman projections for regularized transportation problems," *SIAM Journal on Scientific Computing*, vol. 37, no. 2, pp. A1111–A1138, 2015.

[22] D. Zhu, P. Cui, D. Wang, and W. Zhu, "Deep variational network embedding in wasserstein space," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 2827–2836.

[23] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, "Wasserstein weisfeiler–lehman graph kernels," in *Advances in Neural Information Processing Systems 32 (NeurIPS)*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 6436–6446.

[24] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs." in *NIPS*, 2009, pp. 1660–1668.

[25] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[26] F. Mémoli, "Gromov–wasserstein distances and the metric approach to object matching," *Foundations of computational mathematics*, vol. 11, no. 4, pp. 417–487, 2011.

[27] G. Peyré, M. Cuturi, and J. Solomon, "Gromov-wasserstein averaging of kernel and distance matrices," in *International Conference on Machine Learning*. PMLR, 2016, pp. 2664–2672.

[28] V. Titouan, N. Courty, R. Tavenard, and R. Flamary, "Optimal transport for structured data with application on graphs," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6275–6284.

[29] H. Xu, D. Luo, H. Zha, and L. C. Duke, "Gromov-wasserstein learning for graph matching and node embedding," in *International conference on machine learning*. PMLR, 2019, pp. 6932–6941.

[30] H. Xu, D. Luo, and L. Carin, "Scalable gromov-wasserstein learning for graph partitioning and matching," *Advances in neural information processing systems*, vol. 32, pp. 3052–3062, 2019.

[31] H. Xu, "Gromov-wasserstein factorization models for graph clustering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 6478–6485.

[32] H. Wang and A. Banerjee, "Bregman alternating direction method of multipliers," *Advances in Neural Information Processing Systems*, vol. 4, no. January, pp. 2816–2824, 2014.

[33] C. Vincent-Cuaz, T. Vayer, R. Flamary, M. Corneli, and N. Courty, "Online graph dictionary learning," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 10 564–10 574.

[34] L. Chen, Z. Gan, Y. Cheng, L. Li, L. Carin, and J. Liu, "Graph optimal transport for cross-domain alignment," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1542–1553.

[35] H. P. Maretic, M. El Gheche, G. Chierchia, and P. Frossard, "Got: an optimal transport framework for graph comparison," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 13 876–13 887.

[36] Y. Dong and W. Sawin, "Copt: Coordinated optimal transport on graphs," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[37] T. Ma and J. Chen, "Unsupervised learning of graph hierarchical abstractions with differentiable coarsening and optimal transport," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 8856–8864.

[38] J. W. Ruge and K. Stüben, "Algebraic multigrid," in *Multigrid methods*. SIAM, 1987, pp. 73–130.

[39] J. Klicpera, M. Lienen, and S. Günnemann, "Scalable optimal transport in high dimensions for graph distances, embedding alignment, and more," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5616–5627.

[40] A. Bojchevski and S. Günnemann, "Certifiable robustness to graph perturbations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 8317–8328.

[41] A. Dalmia and M. Gupta, "Towards interpretation of node embeddings," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 945–952.

[42] N. Liu, X. Huang, J. Li, and X. Hu, "On interpretation of network embedding via taxonomy induction," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1812–1820.

[43] N. Bonneel, G. Peyré, and M. Cuturi, "Wasserstein barycentric coordinates: histogram regression using optimal transport." *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 71–1, 2016.

[44] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," *arXiv preprint arXiv:2005.03675*, 2020.

[45] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[46] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[47] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in Neural Information Processing Systems (NIPS)*, 2002, pp. 585–591.

[48] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 37–48.

[49] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1105–1114.

[50] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, pp. 891–900.

[51] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks, "A closer look at skip-gram modelling." in *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, vol. 6, 2006, pp. 1222–1225.

[52] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.

[53] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 855–864.

[54] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[55] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 1067–1077.

[56] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1225–1234.

[57] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *AAAI Conference on Artificial Intelligence*, 2016.

[58] A. Blum, T. H. Chan, and M. R. Rwebangira, "A random-surfer web-graph model," in *2006 Proceedings of the Third Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. SIAM, 2006, pp. 238–246.

[59] M. A. Schmitz, M. Heitz, N. Bonneel, F. Ngole, D. Coeurjolly, M. Cuturi, G. Peyré, and J.-L. Starck, "Wasserstein dictionary learning: Optimal transport-based unsupervised nonlinear dictionary learning," *SIAM Journal on Imaging Sciences*, vol. 11, no. 1, pp. 643–678, 2018.

[60] J. Bouttier, P. Di Francesco, and E. Guitter, "Geodesic distance in planar graphs," *Nuclear physics B*, vol. 663, no. 3, pp. 535–567, 2003.

[61] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[62] A. Kroshnin, N. Tupitsa, D. Dvinskikh, P. Dvurechensky, A. Gasnikov, and C. Uribe, "On the complexity of approximating wasserstein barycenters," in *International Conference on Machine Learning (ICML)*, 2019, pp. 3530–3540.

[63] M. Defferrard, L. Martin, R. Pena, and N. Perraudin, "Pygsp: Graph signal processing in python." [Online]. Available: https://github.com/epfl-lts2/pygsp/

[64] E. Simou and P. Frossard, "Graph Signal Representation with Wasserstein Barycenters," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 5386–5390.

[65] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[66] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop on Bayesian Deep Learning*, 2016.

[67] ——, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[68] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

[69] *OrgNet*, 2004). [Online]. Available: http://www.orgnet.com/divided2.html

[70] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI Conference on Artificial Intelligence*, 2015. [Online]. Available: http://networkrepository.com

[71] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR (Poster)*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[72] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social networks*, vol. 5, no. 2, pp. 109–137, 1983.

[73] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *The Journal of Machine Learning Research*, vol. 11, pp. 2837–2854, 2010.

[74] M. Blondel, Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, F. Pedregosa, and J.-P. Vert, "Efficient and modular implicit differentiation," *arXiv e-prints*, pp. arXiv–2105, 2021.

[75] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[76] H. Gao and S. Ji, "Graph u-nets," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 2083–2092.

[77] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 3734–3743.

[78] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[79] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in *International Conference on Learning Representations (ICLR)*, 2020.

[80] Y. G. Wang, M. Li, Z. Ma, G. Montúfar, X. Zhuang, and Y. Fan, "Haar graph pooling," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

[81] A. Haar, "Zur theorie der orthogonalen funktionensysteme," *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, 1910.

[82] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[83] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*. ACM, 2020, pp. 2729–2738.

[84] G. Dantzig and D. R. Fulkerson, "On the max flow min cut theorem of networks," *Linear inequalities and related systems*, vol. 38, pp. 225–231, 2003.

[85] D. Mesquita, A. Souza, and S. Kaski, "Rethinking pooling in graph neural networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[86] S. Kolouri, N. Naderializadeh, G. K. Rohde, and H. Hoffmann, "Wasserstein embedding for graph learning," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=AAes_3W-2z

[87] L. Ambrosio, N. Gigli, and G. Savaré, *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2008.

[88] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer, "Pot: Python optimal transport," *Journal of Machine Learning Research*, vol. 22, no. 78, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-451.html

[89] G. Mialon, D. Chen, A. d'Aspremont, and J. Mairal, "A trainable optimal transport embedding for feature aggregation and its relationship to attention," in *ICLR 2021-The Ninth International Conference on Learning Representations*, 2021.

[90] B. Schölkopf, A. J. Smola, F. Bach *et al.*, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[91] S. G. Krantz and H. R. Parks, *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2012.

[92] "Ott toolbox," https://ott-jax.readthedocs.io/en/latest/, released:March 2021.

[93] M. Cuturi, O. Teboul, J. Niles-Weed, and J.-P. Vert, "Supervised quantile normalization for low rank matrix factorization," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2269–2279.

[94] F. Zhang, *The Schur complement and its applications*. Springer Science & Business Media, 2006, vol. 4.

[95] E. Stiefel, "Methods of conjugate gradients for solving linear systems," *J. Res. Nat. Bur. Standards*, vol. 49, pp. 409–435, 1952.

[96] S. Afriat, "Theory of maxima and the method of lagrange," *SIAM Journal on Applied Mathematics*, vol. 20, no. 3, pp. 343–357, 1971.

[97] A. Takayama and T. Akira, *Mathematical economics*. Cambridge university press, 1985.

[98] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[99] J. J. Sutherland, L. A. O'brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships," *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1906–1915, 2003.

[100] G. King and L. Zeng, "Logistic regression in rare events data," *Political analysis*, vol. 9, no. 2, pp. 137–163, 2001.

[101] J. Feydy, T. Séjourné, F.-X. Vialard, S.-i. Amari, A. Trouve, and G. Peyré, "Interpolating between optimal transport and mmd using sinkhorn divergences," in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and M. Sugiyama, Eds., vol. 89. PMLR, 16–18 Apr 2019, pp. 2681–2690. [Online]. Available: http://proceedings.mlr.press/v89/feydy19a.html

[102] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[103] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[104] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[105] D. Pollard, "Quantization and the method of k-means," *IEEE Transactions on Information theory*, vol. 28, no. 2, pp. 199–205, 1982.

[106] G. D. Canas and L. A. Rosasco, "Learning probability measures with respect to optimal transport metrics," in *Proceedings of the 25th International Conference on Neural Information Processing Systems-Volume 2*, 2012, pp. 2492–2500.

[107] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[108] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," in *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. [Online]. Available: www.graphlearning.io

[109] J. Godwin*, T. Keck*, P. Battaglia, V. Bapst, T. Kipf, Y. Li, K. Stachenfeld, P. Veličković, and A. Sanchez-Gonzalez, "Jraph: A library for graph neural networks in jax." 2020. [Online]. Available: http://github.com/deepmind/jraph

[110] M. Hessel, D. Budden, F. Viola, M. Rosca, E. Sezener, and T. Hennigan, "Optax: composable gradient transformation and optimisation, in jax!" 2020. [Online]. Available: http://github.com/deepmind/optax

[111] T. Hennigan, T. Cai, T. Norman, and I. Babuschkin, "Haiku: Sonnet for JAX," 2020. [Online]. Available: http://github.com/deepmind/dm-haiku

[112] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[113] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[114] M. Cuturi and D. Avis, "Ground metric learning," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 533–564, 2014.

[115] M. Heitz, N. Bonneel, D. Coeurjolly, M. Cuturi, and G. Peyré, "Ground metric learning on graphs," *Journal of Mathematical Imaging and Vision*, vol. 63, no. 1, pp. 89–107, 2021.

[116] F. Wang and L. J. Guibas, "Supervised earth mover's distance learning and its computer vision applications," in *European Conference on Computer Vision*. Springer, 2012, pp. 442–455.

[117] G. Zen, E. Ricci, and N. Sebe, "Simultaneous ground metric learning and matrix factorization with earth mover's distance," in *2014 22nd International Conference on Pattern Recognition*. IEEE, 2014, pp. 3690–3695.

[118] V. Vapnik, *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

Effrosyni **Simou**

Machine Learning Engineer

ersisimou.github.io
github.com/ersisimou
@ effrosyni.simou@epfl.ch

Education

Jul 2016 – Oct 2021 | PhD, Signal Processing Laboratory- LTS4, École Polytechnique Federale de Lausanne, EPFL
- Supervisor : Prof. Pascal Frossard
- Title : Graph Representation Learning with Optimal Transport : Analysis and Applications
- Examiners : Prof. P. Vandergheynst, Prof. X. Bresson, Dr. J. Feydy, President : Prof. J-Ph. Thiran
- Thesis successfully defended on October 15th 2021

Sep 2013 – Sep 2014 | MSc Communications and Signal Processing, Imperial College London, ICL
- Graduated with Merit
- MSc thesis project ranked among the three best

Oct 2007 – Jul 2013 | Dipl. Electrical Engineering and Computer Science, National Technical University of Athens, NTUA
- Graduated with Distinction
- Five year Diploma equivalent to MEng
- Specialization stream : Communications and Signal Processing

Experience

Jul 2016 – Oct 2021 | Doctoral Assistant, LTS4 Laboratory, École Polytechnique Federale de Lausanne, EPFL
- Designed, developed and evaluated an autoencoder with a Wasserstein barycentric decoder for graph structured data. The autoencoder leads to interpretable node embeddings that are stable to perturbations of the graph structure and exhibits an increase in performance in node classification tasks compared to state-of-the-art algorithms.
  Python | PyTorch | Matplotlib | scikit-learn | GitHub
- Designed, developed and evaluated a global pooling layer for Graph Neural Network architectures. The pooling layer optimally preserves the statistics by minimizing the Wassesrtein distance between a graph's representation and its pooled version and leads to an increase in graph classification accuracy compared to state-of-the-art methods.
  Python | JAX | Flax | Optax | Jraph | OTT | Matplotlib | scikit-learn | GitHub
- Contributed to the development and the evaluation of a continuous domain adaptation method for temporally evolving data using Optimal Transport. The method demonstrated an increase in classification accuracy with respect to competing methods.
  Python | Matplotlib | scikit-learn | GitHub
- Supervised projects of master students on Graph Neural Network architectures.

Sep 2017 – Feb 2020 | Teaching Assistant, École Polytechnique Federale de Lausanne, EPFL
A Network Tour of Data Science (NTDS) : Master's course. Created, demonstrated and evaluated practical sessions of the course. The practical sessions focused on the collection, analysis and visualization of network-structured data, as well as on their exploitation with machine learning algorithms.
Python | NumPy | Matplotlib | scikit-learn | pandas | GitHub

Mar 2016 – Jun 2016 | Research Intern, LTS4 Laboratory, École Polytechnique Federale de Lausanne, EPFL
Performed an experimental analysis in order to quantify the progressive miss-representation of adversarial images at the hidden layers of Deep Neural Network classifiers.
Matlab | MatConvNet

Apr 2015 – Aug 2015 | Research Assistant | Scientific Programmer, IBUG Laboratory, Imperial College London, ICL
Contributed to the annotation of data within the framework of development of the Menpo project (www.menpo.org) for deformable modelling. Annotation was performed both manually as well as with the use of trained models.
Python | NumPy | Matplotlib | scikit-learn | pandas | GitHub

Jan 2010 – Sep 2011 | Assistant, Residential Investments S.A., Athens, Greece
Part-time. Administrative and technical duties related to building construction and development.

Leadership and Awards

Oct 2016 – Oct 2018 | **PhD Student Representative, Doctoral School of Electrical Engineering (EDEE), EPFL**
Elected from the EDEE student body to represent them in the doctoral program committee meetings and the doctoral commission meetings, which are chaired by the Vice-President for Education

Oct 2014 | **Kontaxi Award, National Technical University of Athens, NTUA**
Honorary award for ranking first (with GPA= 9/10) among the 2013 graduates of the NTUA Department of Electrical Engineering and Computer Science within the stream of Communications and Signal Processing.

Mar 2014 | **IKY Award, State Scholarship Foundation**
Honorary award for obtaining the highest overall grade (10/10) in the academic year 2011/2012 among the students in the NTUA Department of Electrical Engineering and Computer Science.

Jul 2013 | **Thomaidio Prize, National Technical University of Athens, NTUA**
Monetary prize for achieving the highest overall grade (10/10) in the academic year 2011/2012 among all students from all years in the NTUA Department of Electrical Engineering and Computer Science.

Sep 2007 | **Outstanding performance in the Greek National Exams**
Awarded for ranking among the top 10 % of the students with the best performance on the Greek National Exams that were admitted to the NTUA Department of Electrical Engineering and Computer Science .

Skills

Programming Languages	Python, C++
Deep learning frameworks	JAX, PyTorch, TensorFlow, MatConvNet
Scientific Computing	Python (NumPy, Matplotlib, scikit-learn, pandas), Matlab
Open Source	Familiarity with open-source GitHub development and management workflows.

Languages

- **Greek** : mother tongue
- **English** : Fluent, Certificate of Proficiency in English, University of Cambridge, Grade obtained : A
- **French** : Advanced, Level B2, Institut Francais d Athènes

Publications

PhD Thesis

- **Graph Representation Learning with Optimal Transport : Analysis and Applications**, Effrosyni Simou, *October 2021.*

Journal

- **node2coords : Graph Representation Learning with Wasserstein Barycenters**, Effrosyni Simou, Dorina Thanou, Pascal Frossard, *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 17–29, 2020.

Conference

- **Forward-Backward Splitting for Optimal Transport Based Problems**, Guillermo Ortiz-Jiménez, Mireille El Gheche, Effrosyni Simou, Hermina Petric Maretić, Pascal Frossard, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.
- **Graph Signal Representation with Wasserstein Barycenters**, Effrosyni Simou and Pascal Frossard, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

Workshop

- **CDOT : Continuous Domain Adaptation using Optimal Transport**, Guillermo Ortiz-Jiménez, Mireille El Gheche, Effrosyni Simou, Hermina Petric Maretić, Pascal Frossard, *Optimal Transport & Machine learning (OTML) Workshop at 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2019.