Graph learning with bilevel optimization

Hashem Ghanem

UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

INSTITUT DE MATHÉMATIQUES
DE BOURGOGNE

THÈSE DE DOCTORAT DE L'ÉTABLISSEMENT

UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

PRÉPARÉE À L'INSTITUT DE MATHÉMATIQUES DE BOURGOGNE

École doctorale n° 553 : Carnot-Pasteur

Graph learning with bilevel optimization

Apprentissage de graphes via l'optimisation bi-niveau

THÈSE

Pour l'obtention du titre de

DOCTEUR EN SCIENCES

SPÉCIALITÉ MATHÉMATIQUES APPLIQUÉES

Présentée par

Hashem GHANEM

Soutenue le 12 septembre 2023 devant le jury composé de

Pierre-Olivier AMBLARD	CNRS, GIPSA-Lab, Université Grenoble-Alpes	Rapporteur
Hervé CARDOT (président du jury)	IMB, Université de Bourgogne	Examinateur
Nicolas KERIVEN	CNRS, IRISA, Université de Rennes	Co-encadrant
Nicolas PAPADAKIS	CNRS, IMB, Université de Bordeaux	Rapporteur
Barbara PASCAL	CNRS, LS2N, Université de Nantes	Examinatrice
Patricia REYNAUD-BOURET	CNRS, LJAD, Université Côte d'Azur	Examinatrice
Joseph SALMON	IMAG, Université de Montpellier	Co-directeur
Samuel VAITER	CNRS, LJAD, Université Côte d'Azur	Directeur

UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

INSTITUT DE MATHÉMATIQUES
DE BOURGOGNE

THÈSE DE DOCTORAT DE L'ÉTABLISSEMENT

UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

PRÉPARÉE À L'INSTITUT DE MATHÉMATIQUES DE BOURGOGNE

École doctorale n° 553 : Carnot-Pasteur

Graph learning with bilevel optimization

Apprentissage de graphes via l'optimisation bi-niveau

THÈSE

Pour l'obtention du titre de

DOCTEUR EN SCIENCES

SPÉCIALITÉ MATHÉMATIQUES APPLIQUÉES

Présentée par

Hashem GHANEM

Soutenue le 12 septembre 2023 devant le jury composé de

Pierre-Olivier AMBLARD	CNRS, GIPSA-Lab, Université Grenoble-Alpes	Rapporteur
Hervé CARDOT (président du jury)	IMB, Université de Bourgogne	Examinateur
Nicolas KERIVEN	CNRS, IRISA, Université de Rennes	Co-encadrant
Nicolas PAPADAKIS	CNRS, IMB, Université de Bordeaux	Rapporteur
Barbara PASCAL	CNRS, LS2N, Université de Nantes	Examinatrice
Patricia REYNAUD-BOURET	CNRS, LJAD, Université Côte d'Azur	Examinatrice
Joseph SALMON	IMAG, Université de Montpellier	Co-directeur
Samuel VAITER	CNRS, LJAD, Université Côte d'Azur	Directeur

Abstract

This thesis focuses on *graph learning* for *semi-supervised learning* tasks to mitigate the impact of noise in real-world graphs. One approach to learn graphs is using *bilevel optimization*, whose inner problem optimizes the downstream model, and its outer problem evaluates the performance of the optimized model *w.r.t.* a labelling loss and updates the graph accordingly. This problem is intractable in general. One solution is replacing the inner optimizer by the output of an iterative algorithm converging to a good proxy, and then employing *automatic differentiation* to evaluate its derivative *w.r.t.* the graph, which is learned using a gradient-based algorithm. In this thesis, we first propose to apply this approach to learn analysis-sparsity priors, which boils down to a graph learning problem in applications related to graph Total Variation. Although the problem is non-smooth, we empirically prove the capacity of this solver in 1D and 2D signal denoising tasks. We then propose to use bilevel optimization to train a parametric model on predicting similarity between nodes, instead of learning the graph directly. We show that this notably improves performance over observed graphs. Finally, we identify and analyze the *gradient scarcity* problem, which consists in a lack of supervision on edges connecting distant unlabelled nodes. We prove that this issue emerges when directly optimizing the observed edges while using graph neural networks or the Laplacian regularization in the downstream task. We examine several solutions to this issue including metric learning, graph regularization, or expanding the graph, and prove their efficiency.

Keywords: gradient scarcity, graph learning, bilevel optimization, semi-supervised learning, automatic differentiation.

Résumé

Cette thèse se concentre sur l'*apprentissage de graphes* pour les tâches d'*apprentissage semi-supervisé* afin d'atténuer l'impact du bruit dans les graphes du monde réel. Une approche pour apprendre les graphes est d'utiliser l'*optimisation bi-niveau*, dont le problème interne optimise le modèle en aval, et son problème externe évalue la performance du modèle optimisé par rapport à une fonction de perte d'étiquetage et met à jour le graphe en conséquence. Ce problème est en général numériquement intractable. Une solution consiste à remplacer l'optimiseur interne par la sortie d'un algorithme itératif convergeant vers un bon proxy, puis à utiliser la *différentiation automatique* pour évaluer sa dérivée par rapport au graphe, qui est appris à l'aide d'un algorithme basé sur le gradient. Dans cette thèse, nous proposons d'abord d'appliquer cette approche pour apprendre les priorités d'analyse-parcimonie, ce qui revient à un problème d'apprentissage de graphe dans les applications liées à la variation totale de graphe. Bien que le problème soit non-lisse, nous prouvons empiriquement la capacité de ce solveur dans les tâches de débruitage de signaux 1D et 2D. Nous proposons ensuite d'utiliser l'optimisation bi-niveau pour entraîner un modèle paramétrique sur la prédiction de la similitude entre les nœuds, au lieu d'apprendre directement le graphe. Nous montrons que cela améliore notablement les performances par rapport aux graphes observés. Enfin, nous identifions et analysons le problème de *gradient scarcity*, qui consiste en un manque de supervision sur les arêtes reliant des nœuds non étiquetés éloignés. Nous prouvons que ce problème émerge lors de l'optimisation directe des arêtes observées tout en utilisant des réseaux de neurones graphiques ou la régularisation laplacienne dans la tâche en aval. Nous examinons plusieurs solutions à ce problème, notamment l'apprentissage métrique, la régularisation de graphe ou l'expansion du graphe, et prouvons leur efficacité.

Mots clés : gradient scarcity, apprentissage de graphes, optimisation bi-niveau, apprentissage semi-supervisé, différentiation automatique.

Acknowledgements

I would like to express my sincere gratitude to my three supervisors for their support and for what I have learned from them. Thank you Samuel for the considerable time you assigned to guide me through this thesis. I appreciate all the unscheduled meetings during these three years, you being concerned with my future, and the advice you gave me in that context. Thank you Nicolas for your guidance and your limitless availability that made distance not a problem. I have learned a lot from you and I have always been impressed by your insightful ideas. Thank you Joseph for your guidance and for teaching me the importance of following the community conventions in writing and coding. I enjoyed that a lot.

I would like to thank Pierre-Olivier Amblard and Nicolas Papadakis for accepting to review this thesis. I appreciate the time you assigned to read the manuscript and your insightful comments and suggestions to improve it. I also would like to thank Hervé Cardot, Barbara Pascal, Patricia Reynaud-Bouret for accepting to be part of the jury that assesses this work.

I would like to thank Côte d'Azur University and the Laboratoire Jean Alexandre Dieudonné (LJAD) for hosting me during the last two years of my PhD. I would like to thank Magali Crochot in L'Institut de Mathématiques de Bourgogne (IMB) for her time and help in the administrative procedures.

During the first year, I met great people in Dijon whom I admire a lot. I would like to thank the dear flatmates I had there, Alexis and Sacha. Our discussions are valuable memories that I won't forget. My friends at L'Institut de Mathématiques de Bourgogne (IMB), Eddy Oscar Polina and Ioannis, thank you for the great time we spent exchanging our experiences and thoughts which helped me grow, and for the fun we had together in our many activities.

I also would like to thank my friends that I met in Nice. Thank you Juliette and Maëlle for the great two years at our nice collocation. Beside teaching me french, you provided me with support. Thank you Zakaria, Victor, and Antoine for the nice time we spent together at our office and for the help you provided whenever

I needed it. I would like to thank Charbel, Dahmane, Lamine and Najwa as well. I appreciate the deep conversations and the many experiences we shared together which helped me evolve and stay focused. Last but not least, thank you Feras, Ghadeer, Giulia, Obaida, Omar and Reda for being such close friends, and for providing me with positive energy and useful suggestions.

I would like to thank my family in Syria, Fatima, Mohammed, Sajed, Hadi, Yasmeen, Abed, Noura, Sanaa, Nabeel, Muna, Saja, Doaa, Miaz, my aunts, my uncles and my grandparents for their encouragement and for pushing me to improve myself and be responsible for my projects; to be the person who I am now. Finally I would like to thank my parents for believing in me and for motivating me to keep moving toward my goals. Thank you for the efforts you made to provide me with good education. Thank you for your endless support and love that was important to come this far and will be important to go further.

Contents

Chapter 1

Introduction

Graph structures are important tools to model relations and interactions between data points. The need for these structures can be traced back to 1679, when G.W. Leibniz wrote to C. Huygens about the limitations of the traditional coordinate geometry treatment of geometric figures saying "we need yet another kind of analysis, geometric or linear, which deals directly with position, as algebra deals with magnitude", (Tutte and Tutte, 2001). Following this, Leibniz started investigating an alternative tool that he referred to as "geometry of positions" (geometria situs). This geometry, as L. Euler stated in his 1736 renowned work on the Königsberg Bridges problem, "is concerned only with the determination of position, and its properties; it does not involve measurements nor calculations made with them", (Euler, 1741). Euler's work, recognized to mark the beginning of graph theory, and the ones that followed induced a rich literature on graph structures.

Indeed, graphs are used to model networks of interacting entities in many domains. In social networks, graphs model connections between users (Newman et al., 2002). In biology, it is deployed to model chemical interactions between molecules (Stelzl et al., 2005). In transportation, it is used to model the connections between cities (Yu et al., 2017). In these domains, exploiting relations between points when solving the downstream task is important as, take social networks as example, the network members share information resulting in some members changing their beliefs based on the information they receive.

One of the main applications where graphs play a significant role is *Semi-Supervised Learning* (SSL), where only a small fraction of data points have labels and the goal is to predict labels on the remaining unlabeled points. In this setting, graphs can capture similarities and relations between data points and be used to propagate the label information from labeled points to unlabeled ones. In fact, many *graph-based*

semi-supervised learning methods have been developed to incorporate graphs when solving SSL tasks. However, the quality and availability of graphs can significantly affect the performance of these methods. Therefore, there is a need to learn high-quality graphs from data as real-world graphs are inherently noisy or even not given.

In this thesis, we consider the *graph learning* problem for semi-supervised learning tasks, which we address by optimizing graphs as to improve performance in the downstream application. This methodology results in a hierarchical optimization that is called *bilevel optimization*. In the presented material, we address the following questions:

1. how can the aforementioned bilevel optimization be effectively used to learn better graphs for semi-supervised learning tasks?
2. what issues arise when using this methodology? and how to address them?

1.1 Partially labeled datasets are ubiquitous

The amount of generated data has been growing exponentially with the arrival of big data. One of the crucial problems is the extremely expensive cost for labeling all of it. Indeed, generating unlabeled data (features) is easier due to the availability of sensory systems which acquire observations non-stop. In addition, we have the Internet that is a huge sink of data of all types from different users worldwide. However, labels are costly to obtain, as in most situations they are not parameters that can be measured or easily calculated by a computer program, *i.e.,* we do not have access to the label generation process. For example, labeling medical images requires expert knowledge and can be a time-consuming task. Similarly, labeling text data can be a subjective task and may require multiple experts to reach a consensus. In computer vision, labeling images or videos can be labor-intensive and costly. Moreover, it may be impossible in some domains to obtain fully labeled datasets. For instance, not all users in social networks provide the information necessary to label them, and such information has to be inferred from the available data.

Given the massive amount of data, it is clearly impossible to afford either the time cost or the human resources to label all of it. As a result, it is common to observe both labeled and unlabeled data points, the latter being usually the vast majority. Learning tasks on datasets which comprise both labeled and unlabeled points is referred to as *Semi-Supervised Learning* (SSL).

Various SSL methods have been developed to alleviate label scarcity, which makes

predicting labels on unlabeled points more challenging. An early method, called self-training, consists in training a model on the labeled data, then using its most confident predictions on unlabeled points to augment the labeled data (Yarowsky, 1995). This is repeated for many rounds. As an extension of self-training, co-training trains two models on different views of the data to mitigate the bias resulted from augmenting the labeled data with incorrect predictions. Co-training iterates over both models, trains a model on the labeled data in each iteration, then uses its confident predictions to augment the labeled data (Blum and Mitchell, 1998).

Another main approach is to incorporate extra assumptions on the data, which provide additional knowledge to guide the learning process. The main one, called *homophily*, refers to the fact that "nearby" points are likely to have similar labels (Wang and Zhang, 2006). Moreover, points in many applications represent entities that are naturally linked to each other, *e.g.,* in biology (Liu et al., 2018) or social media (Liben-Nowell and Kleinberg, 2003). There again, linked entities are likely to share the same label, which underlines the importance of exploiting the links when solving SSL problems. A natural tool to model these relations is *graph structures*. In fact, graph structures enable a class of methods called *graph-based semi-supervised learning*, which leverage the graph structure and the labeled data to propagate label information to unlabeled points. Thanks to its good performance, these methods have gained significant attention, especially with the recent advances in Graph Neural Networks (GNNs), which can learn more sophisticated interpretations of relations beyond the homophily assumption and have shown promising performance on various SSL tasks. We introduce graph structures in the next section and follow it by a review of the graph-based SSL methods, including homophily-based and GNN-based methods.

1.2 Graph structures

Graph structures are used to model a set of objects and their relations/interactions with each other. While the nature of these objects and their interactions vary with the application, the underlying modeling paradigm is the same for all applications: objects are represented by nodes, and a relation between two objects is represented by an edge between the corresponding two nodes. For instance, in a social network like Facebook, nodes are users and edges are friendships between them. In a biological network such as the brain, nodes are brain regions and edges are the nerve connections in between. See Table 1.1 for a list of different examples. Also refer to Chami et al. (2022); Kazemi et al. (2020).

Notations and definitions: a graph $\mathcal{G}$ is a pair (V, E), where V is a set of n

Network	Nodes	Node features	Edges	Edge features
Transportation system	Cities	Registered cars	Routes	Length, cost
Banking network	Account holders	Account status	Transactions	Transaction value
Social network	Users	Name, country	Interactions	Type (like, comment)

Table 1.1 – Examples of real-world graphs.

nodes and $E \subseteq V \times V$ is a set of edges. Any edge (i, i) is called a self loop. In general, two vertices i and j can be connected by more than one edge. A simple graph is a graph with no self loops and no multiple edges. A graph is said to be undirected if edges have no orientation, *i.e.,* if $(i, j) \in E$ then $(j, i) \in E$. Here we consider simple undirected graphs. We denote by $\mathcal{V}(i)$ the set of neighbors of node i. We also denote by $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ the matrix whose rows include the features of corresponding nodes, where p is the number of features. For example, the user features in a social network might include age, address and work. While in citation networks where nodes represent research publications and edges stand for citations, features are a bag of words used to describe the according publication. Similarly, a feature matrix on edges might be given; however, in this work we consider the case where each edge has a scalar attribute that we refer to by the edge weight.

Definition 1.2.1 (Adjacency matrix)**.** The adjacency matrix $\boldsymbol{A}$ is a square matrix of size $n \times n$ that is used to represent a simple graph. Each entry $\boldsymbol{A}_{i,j}$ equals the weight of the edge from node i to node j.

Definition 1.2.2 (Degree matrix)**.** The degree matrix $\boldsymbol{D}$ of a graph is a diagonal matrix with node degrees on the diagonal: $\boldsymbol{D}_{i,i} = \sum_j \boldsymbol{A}_{i,j}$.

Definition 1.2.3 (Graph Laplacian)**.** The unnormalized graph Laplacian is the matrix defined as follows $\boldsymbol{L} = \boldsymbol{L}(\boldsymbol{A}) = \boldsymbol{D} - \boldsymbol{A}$, where $\boldsymbol{D}$ is the according degree matrix. $\boldsymbol{L}$ is symmetric and positive semi-definite for undirected simple graphs. It is usually common to consider other variations that include normalizing the graph Laplacian by means of the degree matrix. One popular variation is $\boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$, where $\boldsymbol{I} \in \mathbb{R}^{n \times n}$ is the identity matrix.

Definition 1.2.4 (Incidence matrix)**.** The incidence matrix $\boldsymbol{C}$ of a graph is a matrix of size $n \times |E|$. For undirected graphs it is defined as follows:

$$\boldsymbol{C}_{i,j} = \begin{cases} 1 & \text{if node } i \text{ is an endpoint of the } j\text{-th edge,} \\ 0 & \text{otherwise,} \end{cases}$$

while for directed graphs it is defined as follows:

$$\boldsymbol{C}_{i,j} = \begin{cases} 1 & \text{if node } i \text{ is the head of the } j\text{-th edge,} \\ -1 & \text{if node } i \text{ is the tail of the } j\text{-th edge,} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 1.2.5 (Path)**.** We say that the sequence of distinct nodes $(u_1, \ldots, u_k)$ is a path in the graph if $(u_i, u_{i+1}) \in E$ for all i in $\{1, \cdots, k-1\}$. The length of the path is the number of edges composing it which is $k-1$.

Definition 1.2.6 (Connected graphs)**.** A connected graph is a graph where there is at least one path connecting every pair of nodes.

Definition 1.2.7 (Hop distance)**.** We say that node i is k-hop from node j if the minimum number of edges forming a path from i to j is k. Similarly, a node i is said to be k-hop from a subset of nodes if the minimum of hop distances to nodes in the subset is k. We further refer to the set of nodes that are at most k-hop from i by its k-hop neighborhood.

1.3 Graph-based Semi-Supervised Learning (SSL)

In this section, we first define the inductive and the transductive settings of SSL. Then, we elaborate on graph-based SSL methods, and point out the main issue we tackle in this thesis, that is, the dependence of these methods on the quality of the input graph. Finally, we present many real-world applications of these methods.

Given a partially labeled dataset, SSL aims at learning a labeling function that can assign labels to both labeled and unlabeled data in a way that optimizes some objective function, such as accuracy. Depending on the scope of the labeling function, SSL methods can be classified into two categories: *inductive* and *transductive*.

Definition 1.3.1 (Inductive SSL)**.** Given a dataset comprising a subset of unlabeled points $\{(x_i)\}_{i=1}^{n_u} \subset \mathcal{X}$ and a subset of labeled points $\{(x_i, y_i)\}_{i=1}^{n_l} \subset \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}, \mathcal{Y}$ are the feature and the label space, respectively, inductive SSL aims at learning a labeling function that is able to generalize to any point $x \in \mathcal{X}$.

Definition 1.3.2 (Transductive SSL)**.** Given a dataset comprising a subset of unlabeled points $\{(x_i)\}_{i=1}^{n_u} \subset \mathcal{X}$ and a subset of labeled points $\{(x_i, y_i)\}_{i=1}^{n_l} \subset \mathcal{X} \times \mathcal{Y}$, transductive SSL aims at learning from this dataset to be able to predict labels only on the unlabeled subset $\{(x_i)\}_{i=1}^{n_u}$ without considering future data.

Consequently, each category uses the unlabeled data in a different manner. Inductive methods employ unlabeled points to improve the generalization ability of the labeling function. For instance, self-training, which is an inductive method, assumes that most confident predictions on unlabeled data are ground-truth with the purpose of overcoming label scarcity and augmenting the training set, ultimately improving the generalization ability of the learned labeling function. On the other hand, transductive methods leverage unlabeled points to directly propagate or infer their labels by relying on some assumptions such as homophily. This explains why transductive methods usually outperform inductive methods on unlabeled points present in the given dataset (Chong et al., 2020).

In this thesis, we look at graph-based SSL methods, which are developed to tackle tasks where data points lie on a graph structure, regardless of whether the graph is observed as in situations where points are naturally linked together, *e.g.,* social networks, or constructed from data. More precisely, we consider the set of methods developed for the transductive setting. In fact, the majority of graph-based SSL methods are transductive in nature (Chong et al., 2020; Song et al., 2022b), where both labeled and unlabeled points appear as nodes in the graph, and edges are used to model relations in between.

Given $(\boldsymbol{X}_{obs}, \mathcal{G}_{obs}, \boldsymbol{Y}_{obs})$, where $\mathcal{G}_{obs}$ is the observed graph, $\boldsymbol{X}_{obs}$ are the observed node features (we will drop the subscript and write $\boldsymbol{X}$ in the rest of the thesis) and $\boldsymbol{Y}_{obs} \in \mathbb{R}^n$ contains the labels of a subset of points at coordinates $i \in V_{tr} \subset V$ and, *e.g.,* not-a-number "NaN" outside of V_{tr}, the goal of transductive graph-based methods is to predict labels on unlabeled nodes while exploiting the graph structure $\mathcal{G}_{obs}$. From now on, we stop mentioning the transductive setting and refer to these methods as graph-based methods for the convenience of the reader. We also refer to SSL tasks involving a graph structure by graph-based SSL tasks.

Early graph-based methods focused on incorporating the homophily assumption, which implies that the labels $\boldsymbol{Y}_i, \boldsymbol{Y}_j$ of nodes i, j that are neighbors in the graph satisfy $\boldsymbol{Y}_i \approx \boldsymbol{Y}_j$. This is achieved by utilizing the graph structure to regularize the learned labels. This family of methods is referred to as graph regularization methods. In this work, we deploy a commonly used representative of these methods, namely the *Laplacian regularization* which will be presented in the next section. A more recent methodology consists in learning node embeddings by incorporating the graph structure. Then, a model is placed on top of the embedding function to predict labels. Methods in this paradigm are referred to as node embedding methods. We review this methodology in Section 1.3.2, with a special focus on Graph Neural Networks (GNNs) employed in this work as a representative method. Refer to Song et al. (2022b,a) for a more comprehensive review on both method categories.

1.3.1 Graph regularization methods

This methodology consists in *propagating* known labels using a *regularization* process. The goal is that the output labels accurately approximate observed labels on labelled nodes while ensuring smoothness on the graph. The regularization is performed by introducing a penalty term to the objective function that evaluates a candidate labeling. That is, learned labels read:

$$\boldsymbol{Y}_{\text{Reg}} \in \underset{\boldsymbol{Y}\in\mathcal{B}}{\arg\min} \frac{1}{|V_{tr}|} \sum_{i\in V_{tr}} \ell(\boldsymbol{Y}_i, (\boldsymbol{Y}_{obs})_i) + \lambda R(\boldsymbol{Y}, \boldsymbol{A}) \ , \tag{1.1}$$

where $\mathcal{B}$ is an admissible set, ℓ is usually a smooth loss function, R is a regularization term, and λ is a balancing parameter. In regression tasks, $\mathcal{B}$ is commonly the space $\mathbb{R}^n$, and ℓ is chosen to be the Mean Squared Error (MSE) defined as:

$$\ell(\boldsymbol{Y}_i, (\boldsymbol{Y}_{obs})_i) = (\boldsymbol{Y}_i - (\boldsymbol{Y}_{obs})_i)^2 \ .$$

Whilst in classification tasks, the i-th element $\boldsymbol{Y}_i$ is not a scalar but rather a vector holding the probability distribution over classes. Formally, $\mathcal{B} = \{\boldsymbol{Y} \in \mathbb{R}^{n\times C} \mid \forall i, \ \sum_{c=1}^{C} \boldsymbol{Y}_{i,c} = 1, \ \forall i, c, \ \boldsymbol{Y}_{i,c} \geq 0\}$, where C is the number of classes. In this case, ℓ is the Categorical Cross Entropy (CCE) loss defined as:

$$\ell(\boldsymbol{Y}_i, (\boldsymbol{Y}_{obs})_i) = -\log(\boldsymbol{Y}_{i,(\boldsymbol{Y}_{obs})_i}) \ .$$

In fact, graph regularization methods usually differ from each other by the choice of the regularization function R. We next review the two most common choices, the Laplacian regularization (Slepcev and Thorpe, 2019; Pang and Cheung, 2017), which is deployed in this thesis, and the Label Propagation (LP) model (Zhu and Ghahramani, 2002).

Graph Laplacian regularization: this is a commonly deployed method in this category. This approach proved to have a good performance in many reconstruction problems, *e.g.,* image filtering (Milanfar, 2012). The associated choice of the regularization term R is the following (Slepcev and Thorpe, 2019; Pang and Cheung, 2017):

$$R(\boldsymbol{Y}, \boldsymbol{A}) = \frac{1}{|E|} \sum_{i,j} \boldsymbol{A}_{i,j} \|\boldsymbol{Y}_i - \boldsymbol{Y}_j\|_2^2 \ , \tag{1.2}$$

which can be re-written as follows depending on the downstream task:

$$R(\boldsymbol{Y}, \boldsymbol{A}) = \begin{cases} \frac{1}{|E|} \boldsymbol{Y}^\top \boldsymbol{L} \boldsymbol{Y} & \text{in regression tasks,} \\ \frac{1}{|E|} \sum_{c=1}^{C} (\boldsymbol{Y}_{:,c})^\top \boldsymbol{L} \boldsymbol{Y}_{:,c} & \text{in classification tasks,} \end{cases} \tag{1.3}$$

where we denote by $\boldsymbol{Y}_{:,c}$ the c-th column of $\boldsymbol{Y}$.

Note that here the node features $\boldsymbol{X}$ are not used.

The Label Propagation (LP) model: to compute labels for unlabeled nodes, Zhu and Ghahramani (2002) proposed to look at the graph as a flow network where labels start propagating from labeled nodes through the graph till arriving an equilibrium. Edges here can be seen as pipes that carry labels from one node to another with capacity proportional to the edge weight. Formally, the sought-for labels are obtained by performing a series of the following steps till convergence:

1. compute $\boldsymbol{Y}_{t+1} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{Y}_t$. Remark that $\boldsymbol{D}^{-1}\boldsymbol{A}$ is a well-defined transition matrix, *i.e.,* its elements are non-negative and its rows sum to one.
2. clamp the labels of nodes in V_{tr} to their observed values in $\boldsymbol{Y}_{obs}$.

Zhu and Ghahramani (2002) proved that the LP model converges to a steady-state solution. Moreover, the authors proved that when the graph is connected the solution does not depend on the initialization on unlabeled nodes and is unique. In fact, the solution is the minimizer of Eq. (1.1) with $\ell(\boldsymbol{Y}_i, (\boldsymbol{Y}_{obs})_i) = 0$ if $\boldsymbol{Y}_i = (\boldsymbol{Y}_{obs})_i$ and ∞ otherwise, and $R(\boldsymbol{Y}, \boldsymbol{A})$ being set as defined in Eq. (1.3). Later, different variants of LP have been developed like the modified adsorption (Talukdar and Crammer, 2009) and local and global consistency (Zhou et al., 2003) models.

Critics to graph regularization: despite its popularity and the good performance observed in many tasks, most graph regularization methods have been criticized for not exploiting the rich knowledge encoded in node features. Even the methods that incorporate node features using a model f_W parameterized by weights W to map node features to labels, *i.e.,* the output labels $\boldsymbol{Y}_W(\boldsymbol{X})$ read $\boldsymbol{Y}_W(\boldsymbol{X}) = (f_W(\boldsymbol{X}_i))_{i=1}^n$, and solve for W instead of $\boldsymbol{Y}$ as follows

$$\begin{aligned}&\boldsymbol{Y}_{\text{Reg}} = \boldsymbol{Y}_{W^\star}(\boldsymbol{X}), \text{ where}\\ &W^\star \in \arg\min_W \frac{1}{|V_{tr}|}\sum_{i\in V_{tr}} \ell\Big(\big(\boldsymbol{Y}_W(\boldsymbol{X})\big)_i, (\boldsymbol{Y}_{obs})_i\Big) + \lambda R\big(\boldsymbol{Y}_W(\boldsymbol{X}), \boldsymbol{A}\big)\end{aligned}$$

are also criticized for not taking into account the graph structure when designing the function f_W (Yang et al., 2021). That is, these methods incorporate the graph structure only through the regularization term R under the homophily assumption. In many situations, relations between nodes model more complicated interactions than the likelihood of having the same label (Kipf and Welling, 2017). Last but not least, the challenge of matter in this thesis is the dependence of these methods on the quality of the graph, which tends to be noisy or incomplete in many real-world applications.

1.3.2 Node embedding methods

In machine learning, embedding methods are concerned with learning a low-dimensional representation of high-dimensional data while preserving some proximity measure between points. A class of these methods have been developed for graph-based SSL tasks that involve a graph structure, where two categories can be distinguished: the first one outputs a representation vector for the graph, while the second one outputs a representation vector for each node in the graph. Our focus in this thesis is on the latter category, which is referred to as *node embedding methods*. These methods learn embeddings that capture the graph structure and are informative about the relations of the according node to other nodes in the graph. The produced embeddings are then fed to another model to predict labels on unlabeled nodes. It is noteworthy that this model usually takes vector-based input, *i.e.,* it is not required to handle a graph structure as input.

Two classes of node embedding methods exist, the first one computes node embeddings based on input graphs, while the second one incorporates both the graph structure and node features. The first approach includes factorization-based methods (Ahmed et al., 2013), and random walk-based methods (Grover and Leskovec, 2016; Perozzi et al., 2014). Likewise, these methods do not exploit node features and lack scalability, as they compute an embedding for each node without learning a set of shared parameters. Therefore, we do not consider them in this thesis. The second set of methods, on the other hand, includes autoencoder-based methods, *e.g.,* GAE & VGAE (Kipf and Welling, 2016b) and SDNE (Wang et al., 2016). It also includes Graph Neural Networks (GNNs), which we employ in this thesis.

Graph Neural Networks (GNNs): GNNs are considered to be the modern node embedding framework, which provides state-of-the-art results in many graph SSL tasks. Indeed, thanks to its success and promising results, GNN-based methods are the dominant graph-based methods. GNNs are a class of neural networks that are designed to operate on graphs. The key component is the message passing model implemented in each of the GNN layers, which computes a new embedding of each node based on its features and the features of its neighbor nodes (Gilmer et al., 2017; Wu et al., 2020). Specifically, the output embedding $\boldsymbol{X}_i^{[l]}$ of the l-th layer for node i is computed as follows:

$$\boldsymbol{X}_i^{[l]} = \text{update}\big(\boldsymbol{X}_i^{[l-1]},\ \text{aggregate}(\{\boldsymbol{X}_j^{[l-1]},\ \forall j \in \mathcal{V}(i)\})\big)\ ,$$

where the first layer is fed with $\boldsymbol{X}^{[0]} = \boldsymbol{X}$ in input. That is, we first aggregate the features of the neighbor nodes using a permutation invariant function, *e.g.,* weighted sum using edge weights, and then use the result to update the embedding of the node using a feed-forward neural network for instance. Given that,

different GNN architectures differ in the aggregation and update functions. In this thesis, we adopt the architecture which computes embeddings following the model (Morris et al., 2019):

$$\boldsymbol{X}^{[l]} = \phi(\boldsymbol{X}^{[l-1]}\boldsymbol{W}_1^{[l]} + \boldsymbol{A}\boldsymbol{X}^{[l-1]}\boldsymbol{W}_2^{[l]} + \mathbf{1}_n(\boldsymbol{b}^{[l]})^\top) \ , \tag{1.4}$$

where $\boldsymbol{W}_1^{[l]}, \boldsymbol{W}_2^{[l]} \in \mathbb{R}^{d_{l-1} \times d_l}$ are learnable weights, $\boldsymbol{b}^{[l]} \in \mathbb{R}^{d_l}$ is a learnable bias, d_l is the output dimensionality of the l-th layer, $\mathbf{1}_n = (1, \ldots, 1)^\top \in \mathbb{R}^n$, and ϕ is a non-linear function applied element-wise.

While in classic node embedding methods, predicted labels are obtained by feeding the learned embeddings to another model, in GNN-based methods, it is common to design the GNN and optimize its weights such that its output holds the predicted labels. That is, setting the number of layers k and denoting by W the model weights $W = \{\boldsymbol{W}_1^{[l]}, \boldsymbol{W}_2^{[l]}, \boldsymbol{b}^{[l]}\}_{l=1}^k$, and by $\boldsymbol{Y}_W(\boldsymbol{X}, \boldsymbol{A}) = \boldsymbol{X}^{[k]}$ its output obtained after k rounds of message passing, the sought-for labels read

$$\begin{aligned} &\boldsymbol{Y}_{GNN} = \boldsymbol{Y}_{W^\star}(\boldsymbol{X}, \boldsymbol{A}), \text{ where} \\ &W^\star \in \arg\min_W \frac{1}{|V_{tr}|} \sum_{i \in V_{tr}} \ell\Big(\big(\boldsymbol{Y}_W(\boldsymbol{X}, \boldsymbol{A})\big)_i, (\boldsymbol{Y}_{obs})_i\Big) \ . \end{aligned} \tag{1.5}$$

Drawbacks of GNNs: one issue with GNNs is that its performances significantly declines as labeled nodes get scarce. More importantly, its performances is highly dependent on the graph quality. In this work, we focus on the latter issue.

Remark 1.3.1 (GNNs vs. graph regularization)**.** Comparing the two procedures, graph regularization promotes similarity between connected nodes but, unlike GNNs, is not a supervised-based method capable of learning more complicated schemes of how knowledge propagate through node connections. This is why it generally yields lower performance (Ye et al., 2022).

1.3.3 Real-world applications of graph-based SSL

Graph-based SSL is widely applied in many domains. Some of these domains have the data inherently lying on a graph structure, such as social, citation, and biological networks. In other domains, data points do not exhibit relations between them, and the graph is constructed from scratch, such as in computer vision and natural language processing. In the following, we present a few example applications of graph-based SSL in both scenarios.

Social networks: these networks present numerous problems that can be addressed with graph-based SSL. In Twitter for example, Balaanand et al. (2019)

used graph-based SSL to detect fake users. Another work Alam et al. (2018) employed these methods to classify tweets during natural disasters, which helps to identify the most affected areas and provides fast feedback to humanitarian organizations.

Biology: Graphs are used to model relations in omics data, then graph-based methods are applied. On each omic level, such as chromosomes, DNA, gene expression, microRNA, Kim et al. (2012) construct a graph and then employ a graph-based SSL method to classify cells of glioblastoma multiforme (a tumor that attacks the central nervous system) to low and high grade cells, where different grades in this context reflect the spreading capacity of these cells. Finally, the results of the different omic levels are aggregated to obtain the final classification. To better exploit relations between omic data points in different levels, Doostparast Torshizi and Petzold (2018) make use of extra biological knowledge, known as biological pathways, to construct one graph across these levels, then deploy graph-based SSL to enhance the classification results on tumor cells.

Natural language processing: N-grams are sequences of N words defined to characterize text data, thereby it is important to identify the part of speech of key words in N-grams. This task is referred to as part-of-speech tagging (POS tagging). To solve this task, Subramanya et al. (2010) construct a similarity graph on N-grams, assume that the POS of a word tends to be the same in neighbor N-grams, and then deploy LP to leverage this assumption and predict the POS tagging of words. Similarly, Aliannejadi et al. (2017) solve this problem using GNN models. Recent works make use of graph-based SSL for language model smoothing, which is used to address the sparsity of training data (Mei et al., 2008).

1.4 Why graph learning?

Graph-based learning attracted notably more attention as Graph Convolution Networks (GCNs) showed high performance (Kipf and Welling, 2016a; Monti et al., 2017). Similar to other graph-based methods, GNNs, however, significantly depend on the quality of its input graph (Henaff et al., 2015; Defferrard et al., 2016). Thus, ensuring access to high quality graphs has been a hot topic in research, as real-world graphs are noisy, which significantly degrades performance. Indeed, various methods have been developed for that purpose.

The vast number of developed graph learning methods leads to different possible taxonomies of these methods. In fact, many criteria can be used to classify these methods including:

- the capacity to add/remove edges in the observed graph, where some meth-

ods specialize in optimizing for edge weights in the observed graph, while other methods can also add new edges. We refer to the former as *edge refinement* methods and to the latter as *graph construction* methods. In fact, graph construction methods can be applied to a wider range of tasks, specifically in situations where the observed graph includes missing edges or even not given; however, they are computationally more expensive.

- priors imposed on the learned graph, such as sparsity, low rank graphs, or feature/label smoothness.
- learned variables, where some methods directly learn the adjacency matrix, while others learn the parameters of a graph generative model.
- input to the graph optimization problem, where some methods use node features in addition to the observed graph to learn a better graph, while other methods also incorporate observed labels. We refer to these two categories as
  - feature-based unsupervised graph learning.
  - *label-based supervised graph learning.*

The next sections provide a brief review of the literature of graph learning following the last taxonomy. However, we try to elaborate on the other criteria when possible. Refer to Zhu et al. (2021); Qiao et al. (2018) for a comprehensive review of graph learning methods.

Before proceeding, we point out a common aspect in most graph learning algorithms. They all seek for sparse graphs, *i.e.,* graphs comprising n nodes and $\mathcal{O}(n^2)$ edges. One might think that efforts made to enforce sparsity is only in exchange of providing an efficient algorithm, with respect to both the computational and storage costs. In fact, ensuring sparsity is crucial for additional reasons. First, algorithms outputting sparse graphs are robust against noise (Jebara et al., 2009). Moreover, real-world graphs are indeed sparse, which makes such property useful to solve real-world problems. In addition, sparse graphs are easier to visualize, analyse, and interpret as patterns are more detectable when having fewer edges (Batjargal et al., 2019).

1.4.1 Feature-based unsupervised graph learning

This category includes the majority of graph learning methods, which mainly differs from each other through the hypothesis made on the sought-for graph. Early methods focus on constructing graphs that reflect similarity between nodes, whereas recent methods adopt more sophisticated hypotheses, such as a good

graph is effective in denoising node features with message passing models, feature smoothness, and low rank graphs, to mention just a few. It is interesting to notice that such methods are suitable not only for SSL but for unsupervised problems as well.

Early methods: early methods were developed to construct graphs from scratch. Moreover, the output graph is usually not weighted, *i.e.,* edge weights are binary. One of the most popular algorithms in this context is the k-Nearest Neighbors (k-NN) algorithm. In k-NN, a similarity measure is defined to quantify the strength of the connection, *i.e.,* edge weight, between a pair of nodes. Then, an edge from node i to node j is constructed if j is among the most k similar nodes to i (Zhu et al., 2003; Belkin et al., 2006; Kalofolias and Perraudin, 2017). This method has many drawbacks including the need to choose the value of k and the similarity criterion. The former is usually resolved by means of a validation process to assess different values of k, *e.g.,* using the validation loss in machine learning problems. On the other hand, the similarity measure most of the cases is *a priori* set to one of the classical options, *e.g.,* the Gaussian kernel (Szummer and Jaakkola, 2001), or the cosine similarity. This justifies why data-driven graph learners outperform k-NN graphs. Yet, k-NN graphs are a reliable option and the classical method to initialize graphs for the recent and more advanced learners (Fatemi et al., 2021; Chen et al., 2020). Another shortcoming is the poor scalability as the similarity is evaluated between all possible pairs of points, *i.e.,* $\mathcal{O}(n^2)$ complexity. The common way to alleviate this problem is by using the Approximate Nearest Neighbor (ANN) algorithm that is of cost $\mathcal{O}(n \log(n))$ (Dong et al., 2011; Muja and Lowe, 2014). However, the gain in complexity in ANN comes at the expense of the quality of the constructed graph.

Metric learning methods: a line of works focused on assigning weights to observed edges or to edges constructed using early graph learners discussed above. Recall that edge weights in k-NN graphs are binary. The core idea of this methodology is that edge weights should reflect similarity between nodes, thereby a similarity metric is needed. This metric is either user-defined or learned from data. A popular choice in the former case is the Gaussian kernel coupled with the Euclidean distance as a measure of dissimilarity:

$$\boldsymbol{A}_{ij} = \exp\left(-\frac{\|\boldsymbol{X}_i - \boldsymbol{X}_j\|_2^2}{2\sigma^2}\right) \quad ,$$

where σ is the kernel bandwidth that is usually set using heuristic techniques. For instance, Gretton et al. (2006) use the median distance between nodes. The simplest instance of metric learning is learning the value σ. Zhu et al. (2003) suggested penalizing σ by the average label entropy achieved by the graph-based method on unlabeled nodes. Kapoor et al. (2005) formulate the problem using the

Bayesian framework, where σ is learned by maximizing the marginal likelihood of the observed data. Li et al. (2018) went a step further and proposed a more expressive model by replacing the Euclidean distance with the generalized Mahalanobis distance, which for nodes i, j writes $(\boldsymbol{X}_i - \boldsymbol{X}_j)^\top \boldsymbol{M}\boldsymbol{M}^\top(\boldsymbol{X}_i - \boldsymbol{X}_j)$, where the projecting matrix $\boldsymbol{M} \in \mathbb{R}^{p\times m}$ for some integer m has to be learned.

Locality-inducing methods: most researches in this paradigm optimize edge weights in a given graph by assuming that each node can be produced by a linear combination of its neighbors, where the amplitude of the contribution of each neighbor is the according edge weight. To that end, edges are optimized by minimizing the quadratic error between data points and the aforementioned linear combinations (Saul and Roweis, 2003), *i.e.,* solving the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{A}} \sum_{i=1}^{n} \|\boldsymbol{X}_i - \sum_{j\in\mathcal{V}(i)} \boldsymbol{A}_{ij}\boldsymbol{X}_j\|_2^2 \\ &s.t. \sum_{j\in\mathcal{V}(i)} \boldsymbol{A}_{ij} = 1,\ \forall i \ . \end{aligned}$$

Resulted graphs, however, might have edges with negative weights, which is not desirable for some graph SSL methods, *e.g.,* label propagation models. Accordingly, Wang and Zhang (2006) propose to further constraint edge weights to be non-negative. Interestingly, Daitch et al. (2009) deploy this method to construct the whole adjacency matrix, while theoretically proving that the solution is sparse. This methodology has two main drawbacks, that the quadratic loss is noise sensitive, and that it is prone to overfitting as the number of edges usually is larger than the number of nodes (Qiao et al., 2018).

Smoothness-inducing methods: another line of work focused on learning adjacency matrices that promote smoothness seen in node features on the learned graph. The smoothness is quantified by means of Dirichlet energy:

$$\frac{1}{2}\sum_{i,j} \boldsymbol{A}_{i,j}\|\boldsymbol{X}_i - \boldsymbol{X}_j\|_2^2 \ .$$

Clearly, minimizing this energy *w.r.t.* $\boldsymbol{A}$ will lead to an undesired trivial solution $\boldsymbol{A} = \boldsymbol{0}$. To avoid that, one might add another penalty term to impose a desired structure on the output graph, *e.g.,* distribution of edges or sparsity. Kalofolias (2016) expressed that in following generic optimization:

$$\min_{\boldsymbol{A}} \frac{1}{2}\sum_{i,j} \boldsymbol{A}_{i,j}\|\boldsymbol{X}_i - \boldsymbol{X}_j\|_2^2 + \Upsilon(\boldsymbol{A}) \ ,$$

where the function Υ is designed to promote specific structures in the graph depending on the application. Hu et al. (2013) proposed the following version:

$$\Upsilon(\boldsymbol{A}) = \alpha_1 \|\boldsymbol{A}\mathbf{1}\| + \alpha_2 \|\boldsymbol{A}\|_F^2 + \mathbb{1}(\|\boldsymbol{A}\|_{1,1} = n) \ ,$$

where α_1, α_2 are trade-off parameters, $\|\boldsymbol{A}\|_{1,1} = \sum_{i,j} |\boldsymbol{A}_{ij}|$, and $\mathbb{1}(\text{condition}) = 0$ if the condition is met, and ∞ otherwise. While the third term forces $\boldsymbol{A}$ to sum up to n, the second term tries to divide this sum equally across all edges, and the first one controls the sparsity of the graph by penalizing nodes with large degrees. This choice of Υ produced stat-of-the-art results on graph learning from smooth signals until another version was proposed in Kalofolias (2016):

$$\Upsilon(\boldsymbol{A}) = -\alpha \mathbf{1}^\top \log(\boldsymbol{A}\mathbf{1}) + \beta \|\boldsymbol{A}\|_F^2 \ , \tag{1.6}$$

where the log term prevents producing isolated nodes in the graph, *i.e.,* prevents zero coefficients in the node degrees vector $\boldsymbol{A}\mathbf{1}$. Likewise the k-NN method, this framework costs $\mathcal{O}(n^2)$ and does not scale well with the number of nodes n. Kalofolias and Perraudin (2017) made use of the ANN method to accelerate this framework to $\mathcal{O}(n \log(n))$ with the choice of Υ as in Eq. (1.6) while obtaining quality close to the one produced by the original method.

1.4.2 Label-based supervised graph learning

Our work falls in this category, where one incorporates observed labels in addition to node features to learn graphs. Indeed, this is a natural response to the need to graphs that improve performance in the downstream task. Interestingly, most works in this direction learn together both the graph and the parameters of the graph-based classifier.

The Graph Agreement Model (GAM) is one example that achieved state-of-the-art results in graph-based SSL (Stretcu et al., 2019). The GAM is a deep network trained on predicting similarity between nodes, *i.e.,* metric learning, by penalizing the absence of an edge between nodes with the same label. Once GAM is trained, its output is used to train the adopted GNN model, which is then used to augment the set of labeled nodes by considering confident label predictions. This is repeated for many rounds. The authors provide two versions of their algorithm, one for edge refinement and one for graph construction.

Unlike GAM, Wang and Leskovec (2020) directly learn the weights of observed edges. The optimization also involves learning the parameters of the GNN model. Since this leads to overfitting due to the large number of parameters, the authors make use of the Label Propagation model (LP) to regularize the graph. The

optimization problem is formulated as follows:

$$\min_{\boldsymbol{A},W} \frac{1}{|V_{tr}|} \sum_{i \in V_{tr}} \ell\Big(\big(\boldsymbol{Y}_W(\boldsymbol{X},\boldsymbol{A})\big)_i, (\boldsymbol{Y}_{obs})_i\Big) + \lambda \frac{1}{|V_{tr}|} \sum_{i \in V_{tr}} \ell\Big(\big(\boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A})\big)_i, (\boldsymbol{Y}_{obs})_i\Big) \ ,$$

where $\boldsymbol{Y}_W$ is the output of the GNN model, $\boldsymbol{Y}_{\text{Reg}}$ is the output after τ LP iterations as in Section 1.3.1. The proposed framework produces state-of-the-art results on SSL node classification tasks.

It is important to notice that one objective function is used to jointly assess the graph and the GNN model. We refer to such optimization setting as *joint optimization*. In general, joint optimization is addressed using gradient-based methods. We point out that when computing gradients for these methods, $\boldsymbol{A}$ and W are considered as **independent** variables, *i.e.,* $\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A}) = \boldsymbol{J}_{\boldsymbol{A}}(\boldsymbol{W}) = \boldsymbol{0}$, where $\boldsymbol{W}$ is a vectorized version of W, and $\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A})$ is the Jacobian matrix $\big(\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A})\big)_{i,j} = \frac{\partial(\boldsymbol{W})_i}{\partial \boldsymbol{A}_j}$.

Similarly, Fatemi et al. (2021) regularize the graph by enforcing the assumption that a good graph must also perform well in denoising node features. Consequently, the authors replace the LP-based regularization term above by an objective function that assesses the denoising performance of the graph.

Another sophisticated set of models based on joint optimization is attention mechanisms, where after each GNN layer, the edge weights are re-evaluated based on the similarity between node representations in that layer. The similarity criterion can be user-defined like the dot product (Luong et al., 2015; Vaswani et al., 2017), learned locally at each layer by a single-layer feed-forward network (Veličković et al., 2018), or a combination of both schemes (Kim and Oh, 2021). These mechanisms proved efficient in edge refinement for SSL tasks and some of its variants produced state-of-the-art results. It is noteworthy that in addition to the cheap memory cost of these methods thanks to shared parameters, they generalize well to nodes unseen in training, *i.e.,* the inductive setting.

The class of methods of matter in this thesis assesses a candidate graph by looking at the performance of the graph-based model trained on it. That is to say, the graph-based model is trained while fixing the graph, then the efficiency of the graph is evaluated using another objective function designed for this purpose. To better understand this process, an analogy can be made with the graph being treated as a hyperparameter in a machine learning task, which is typically optimized based on a validation loss function.

One representative work is Franceschi et al. (2019). The authors learn the parameters of Bernoulli probability distributions over independent random edges. These parameters are optimized to minimize the validation loss on a different subset of

nodes than the one used to train the GNN model. This method includes learning n^2 parameters which limits scalability. Similarly, Wan and Kokel (2021) sparsify the observed graph while keeping it connected by means of minimizing the validation loss of the GNN model. However, resulted graphs do not necessarily outperform the observed graph. Moreover, both previous methods suffer from not generalizing to new points, as this requires re-running the optimization process again.

It is important to notice the problem hierarchy in this class of methods. Specifically, evaluating the performance of a candidate graph in the downstream task requires optimizing the graph-based model, whose training process also requires the candidate graph in input. In the next section, we introduce the reader to *bilevel optimization*, which is the mathematical framework that allows us to tackle this problem hierarchy. Simultaneously, we emphasize on the fact that when learning the graph using gradient-based methods, $\boldsymbol{J_W}(\boldsymbol{A}) \neq \mathbf{0}$ and must be incorporated in the gradient computation, unlike the joint optimization scenario. In fact, this is the main challenge in this setting. Then, we present our contribution which mainly includes using bilevel optimization to train a parametric model on node similarity prediction, while mathematically justifying why optimizing edge weights, instead of the parametric model, fails to learn good graphs.

1.5 Bilevel optimization

In this section, we introduce the reader to bilevel optimization. We find that it is convenient to start with real-world examples in order to grasp its importance. Following this, we present the history of bilevel optimization and the first works that tackled it. Next, we provide a formal definition of it and discuss its inherent complexity. We conclude this section by outlining the specific class of bilevel problems that are of interest in this thesis and surveying the existing solutions to these problems.

Optimization problems are commonly single-level, where one optimizes for a set of variables to minimize (or maximize) a single objective function with some constraints. This can be expressed as follows:

$$\begin{aligned} &\min_{\Theta \in \mathcal{A}} F(\Theta) \\ &s.t. \quad G(\Theta) \geq 0 \ , \end{aligned}$$

where F is the objective function, Θ is the set of variables to be optimized for, $\mathcal{A}$ is the admissible set, and G embeds the problem constraints. Indeed, single-level optimization emerges naturally in applications with a *single decision maker* that

controls the parameters of the problem. For example, if a single transportation company designs the timetable of its trips and their stopping points, or if one team of bankers decides on the investments the bank is making.

A different situation takes place in various circumstances in daily life when having two decision makers, where one sets the parameters of the problem while considering the prospective reaction of the other one. This reaction does not only affect the outcome of the latter decision maker, but also the outcome of the former one. That is, any decision made by either side provokes some reward and both sides seek the decision that maximizes their reward. Formalizing this by modelling this reward as a function of the problem variables leads to a hierarchical problem when seen from the point of view of the first decision maker. This is the *bilevel optimization* hierarchy. We tackle a few popular instances of this problem in real world to help better understand the bilevel setting.

The first instance is the toll setting problem, where highway companies tend to assign toll costs to several segments of the highway network as to maximize their revenues. These assignments are made while anticipating the behavior of the network users (drivers) who seek minimizing their traveling cost, which includes toll costs. Therefore, drivers react to any change in toll pricing by varying the frequency of using toll segments in order to travel at minimum cost. For example, if toll costs are set too high, users refrain from using them. Highway companies, on the other hand, take into account this behavior while setting their toll schedule, which boils down to a bilevel optimization problem.

The second example is the pricing problem, where a manufacturer, as the first decision maker, determines the prices of their produced goods to maximize their profits when selling them. Following this, customers, collectively considered as the second decision maker, optimize for the purchased quantity of these items based on their utility-price trade-off. Therefore, the manufacturer's decision depends on the customers optimal response, and conversely, the customers decision is reliant on the pricing decisions of the manufacturer.

Hyper-parameter optimization is also another instance of bilevel optimization. In the context of machine learning, an engineer, as the first decision maker, tunes hyper-parameters to achieve the best validation results in the task in hand. In response to each realization of the hyper-parameters, the optimizer of the machine learning model, as the second decision maker, optimizes for the model weights to obtain optimal results on training data. Vice versa, validation results, which are the engineer's reward, directly depend on the optimized model weights.

1.5.1 History of bilevel optimization

Bilevel optimization has its roots in game theory and economic planning, which can be traced back to the pioneering work of Stackelberg (1952). Stackelberg introduced the concept of leader-follower games, which can be interpreted as a hierarchical game involving two players who interact sequentially. One player, called the leader, is assumed to have complete knowledge of the behavior of the second player, called the follower. The leader makes a move while the follower, who only observes the decisions of the leader, optimizes its own strategies based on that move. The leader anticipates the reactions of the follower and chooses his best strategy to maximize gains. If the follower has multiple optimal responses to a given selection of the leader, the best or worst follower's response *w.r.t.* the leader is assumed, resulting in an optimistic or pessimistic bilevel programming problem, respectively. The optimistic approach is generally assumed; however, it is challenging to ensure this assumption in practice. In fact, when the problem corresponding to the follower game is not convex, it is even difficult to obtain one of the follower's optimal strategies. Instead, obtaining a "good" strategy is considered sufficient. We elaborate on that later on when we discuss the problems of matter in this thesis.

In this hierarchy, we refer to the problem where the leader optimizes its decision as the *outer problem*, and to the problem where the follower does the same, given the leader's decisions, as the *inner problem*.

The first application of the introduced bilevel problem was by Bracken and McGill (1973) on the cost-minimal mix of weapons problem in military. Another one of the first works pioneering bilevel programming problems is Candler and Norton (1977), which started discussing the difficulty of these problems as we will see in the next section. Afterwards, many studies deployed bilevel optimization to model real-world problems of hierarchical structure with two decision makers at two distinct levels. For instance in transportation (Marcotte, 1986; Ben-Ayed et al., 1988), biology and chemistry (Clark, 1990; Sun et al., 2006), management (Bard, 1983; Ryu et al., 2004), and the energy sector (Gabriel et al., 2012). Moreover, it is applied in many machine learning tasks such as hyperparameter optimization, multi-task, and meta-learning (Bennett et al., 2006; Flamary et al., 2014; Muñoz-González et al., 2017; Franceschi et al., 2018), to mention a few. See Colson et al. (2007) for a review of applications in different fields.

1.5.2 General formulation

In mathematical terms, a bilevel optimization problem reads:

$$\begin{aligned} \min_{\Theta \in \mathcal{A}} \quad & F_{out}(\Theta, W^\star(\Theta)) \\ s.t. \quad & G_{out}(\Theta, W^\star(\Theta)) \geq 0 \ , \\ & W^\star(\Theta) \in \arg\min_{W \in \mathcal{B}} F_{in}(W, \Theta) \\ & s.t. \quad G_{in}(W, \Theta) \geq 0 \ , \end{aligned}$$

where Θ is the set of variables controlled by the leader, W is the set of variables adjustable by the follower in response to the leader's decision on Θ, $W^\star(\Theta)$ is the follower's optimal response (or the optimal response corresponding to the optimistic assumption if $W^\star(\Theta)$ is not unique), F_{in} is the inner objective function used by the follower to assess its strategy W, F_{out} is the outer objective function similarly used by the leader to assess decisions on Θ, G_{in}, G_{out} are the inner and outer constraint functions, respectively, and $\mathcal{A}, \mathcal{B}$ are the search spaces according to Θ, W, respectively. Note that F_{in} is parametrized by the leader's assignment to Θ, and that F_{out} anticipates the followers optimal reaction $W^\star(\Theta)$.

It is no doubt that bilevel problems are more complicated and difficult to solve than single-level problems, since the former comprises two nested optimization problems that are dependent. But the question is how difficult is that? Actually Hansen et al. (1992) showed that the simplest instance of bilevel optimization where variables are continuous and all functions are linear is strongly $\mathcal{NP}$-hard. This explains why most works that tackled bilevel optimization looked at its simplest case. Otherwise, different instances of this problem are treated using task-driven methods depending the properties of functions at both levels. That is, every set of properties defines a class of bilevel problems, each class accords to a set of method designed accordingly.

1.5.3 Class of bilevel problems of interest

We consider two instances of bilevel optimization that do not involve the constraint functions G_{in} and G_{out}. Such bilevel problems read:

$$\min_{\Theta \in \mathcal{A}} F_{out}(\Theta, W^\star(\Theta)) \tag{1.7a}$$

$$s.t. \quad W^\star(\Theta) \in \arg\min_{W \in \mathcal{B}} F_{in}(W, \Theta) \ . \tag{1.7b}$$

Unfortunately, in the majority of real-world applications, signal processing and machine learning applications in particular, $W^\star$ does not have a closed form expression as a function of Θ. Hence, Eq. (1.7) cannot be reduced to a single-level

problem, which might be easier to solve. However, many studies focused on obtaining single-level problems using different methodologies. For instance, Hansen et al. (1992); Shi et al. (2005) make use of the optimality conditions of the inner optimization (1.7b) for that purpose. That said, applying this technique results in a single-level problem that includes a lot of constraints, which makes it impractical to implement. Recently, there have been advancements in gradient-based algorithms which are more practical. These algorithms can be grouped into two categories: the Approximate Implicit Differentiation (AID) approach (Domke, 2012; Pedregosa, 2016; Lorraine et al., 2020) and the ITerative Differentiation (ITD) approach (Domke, 2012; Maclaurin et al., 2015; Franceschi et al., 2017). We adopt the latter method in the presented material. Gradient-based methods usually assume that $F_{out}, W^\star$ are continuously differentiable and F_{in} is twice continuously differentiable (Liu et al., 2021a). In order to distinguish between the outer gradient ∇F_{out} and the inner gradient ∇F_{in} in this context, we refer to the outer gradient ∇F_{out} as *hypergradient*.

To correctly express hypergradients, we replace from now on in this section Θ, W by its vectorized version $\boldsymbol{\Theta}, \boldsymbol{W}$, respectively, such that $\boldsymbol{\Theta} \in \mathbb{R}^{d_\Theta}, \boldsymbol{W} \in \mathbb{R}^{d_W}$, where d_Θ, d_W are the number of outer and inner variables, respectively. From the chain rule, the hypergradient at a given pair $(\boldsymbol{\Theta}, \boldsymbol{W}^\star(\boldsymbol{\Theta}))$ writes:

$$\nabla_{\boldsymbol{\Theta}} F_{out} = \frac{\partial F_{out}(\boldsymbol{\Theta}, \boldsymbol{W}^\star(\boldsymbol{\Theta}))}{\partial \boldsymbol{\Theta}} + \boldsymbol{J}_{\boldsymbol{W}^\star}^\top(\boldsymbol{\Theta}) \frac{\partial F_{out}(\boldsymbol{\Theta}, \boldsymbol{W}^\star(\boldsymbol{\Theta}))}{\partial \boldsymbol{W}} \quad . \tag{1.8}$$

The bottleneck when computing hypergradients is evaluating $\boldsymbol{J}_{\boldsymbol{W}^\star}(\boldsymbol{\Theta})$. Recall that in the majority of real-world occurrences of bilevel optimization, $\boldsymbol{W}^\star$ does not enjoy a closed-form expression that can be evaluated.

1.5.4 ITerative Differentiation (ITD)

In this section, we review the first methodology used to evaluate hypergradients. That is, ITerative Differentiation (ITD), which has two main ingredients:

- an iterative algorithm that converges to the solution of the inner problem $\boldsymbol{W}^\star(\boldsymbol{\Theta})$.
- *Automatic Differentiation (AD)* which is capable of differentiating dynamic systems thereby the aforementioned iterative algorithm. A detailed explanation of AD is in Appendix A.

In detail, ITD involves performing τ_{in} updates on the inner problem variables $\boldsymbol{W}$ using the first ingredient, where the dynamics of the t-th update in general are modeled using a continuously differentiable function ψ_t:

$$\boldsymbol{W}_t = \psi_t(\boldsymbol{W}_{t-1}, \boldsymbol{\Theta}), \quad t = 1, 2, \ldots, \tau_{in} \quad ,$$

where $\boldsymbol{W}_0$ is an initialization of $\boldsymbol{W}$ (usually sampled at random). For instance, ψ_t can perform updates of a gradient-based optimizer:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - Q_t(\boldsymbol{W}_t, \nabla_{\boldsymbol{W}_t} F_{in}) \ ,$$

where Q_t is a continuously differentiable function outputting the descent step at iteration t. The simplest example is the gradient descent algorithm where $Q_t(\boldsymbol{W}_t, \nabla_{\boldsymbol{W}_t} F_{in}) = \eta_{in} \nabla_{\boldsymbol{W}_t} F_{in}$, where $\eta_{in} \in \mathbb{R}$ is the according step size, also called the learning rate.

The resulting set of parameters $\boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta})$ is then considered as an approximation of the true solution $\boldsymbol{W}^\star(\boldsymbol{\Theta})$. In other words, ITD replaces the optimal solution $\boldsymbol{W}^\star(\boldsymbol{\Theta})$ with $\boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta})$, hence the bilevel problem of interest (1.7) is replaced by the following problem:

$$\min_{\Theta \in \mathcal{A}} F_{out}(\boldsymbol{\Theta}, \boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta})) \ .$$

With this substitution, the formula of the hypergradient consequently writes:

$$\nabla_{\boldsymbol{\Theta}} F_{out} = \frac{\partial F_{out}\big(\boldsymbol{\Theta}, \boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta})\big)}{\partial \boldsymbol{\Theta}} + \boldsymbol{J}^\top_{\boldsymbol{W}_{\tau_{in}}}(\boldsymbol{\Theta}) \frac{\partial F_{out}(\boldsymbol{\Theta}, \boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta}))}{\partial \boldsymbol{W}} \ .$$

At this point, AD is deployed to differentiate through the iterative updates $\psi_{\tau_{in}} \circ \cdots \circ \psi_2 \circ \psi_1(\boldsymbol{W}_0, \boldsymbol{\Theta})$ and evaluate the term $\boldsymbol{J}^\top_{\boldsymbol{W}_{\tau_{in}}}(\boldsymbol{\Theta})$ and by extension $\nabla_{\boldsymbol{\Theta}} F_{out}$. The process of unfolding inner iterations to differentiate through them is known as *algorithm unrolling*.

Note that machine learning problems in practice are not convex and $\boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta})$ does not necessarily converge towards the optimal point $\boldsymbol{W}^\star(\boldsymbol{\Theta})$, rather does it to a "good" local minima which is considered a good surrogate. Later, we show empirically that in such case replacing $\boldsymbol{W}^\star(\boldsymbol{\Theta})$ by such local minima when learning graph structures provides good results.

1.5.5 Approximate Implicit Differentiation (AID)

The method of interest in this section employs the implicit function theorem to express the hypergradient, then deploys iterative methods to evaluate an *approximation* of it. That is, using the optimality condition in the inner level, which writes

$$\frac{\partial F_{in}(\boldsymbol{W}^\star(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}} = \boldsymbol{0} \ ,$$

and differentiating its both sides with respect to $\boldsymbol{\Theta}$ one gets:

$$\boldsymbol{J}_{\boldsymbol{W}^\star}(\boldsymbol{\Theta}) = -\Big(\frac{\partial^2 F_{in}(\boldsymbol{W}^\star(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \boldsymbol{W}}\Big)^{-1} \frac{\partial^2 F_{in}(\boldsymbol{W}^\star(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \partial \boldsymbol{\Theta}} \ .$$

Substituting into the expression of the hypergradient in Eq. (1.8) we get:

$$\nabla_{\boldsymbol{\Theta}} F_{out} = \frac{\partial F_{out}(\boldsymbol{\Theta}, \boldsymbol{W}^\star(\boldsymbol{\Theta}))}{\partial \boldsymbol{\Theta}} - \Big(\frac{\partial^2 F_{in}(\boldsymbol{W}^\star(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \partial \boldsymbol{\Theta}}\Big)^\top \Big(\frac{\partial^2 F_{in}(\boldsymbol{W}^\star(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \boldsymbol{W}}\Big)^{-1} \frac{\partial F_{out}(\boldsymbol{\Theta}, \boldsymbol{W}^\star(\boldsymbol{\Theta}))}{\partial \boldsymbol{W}} \ . \tag{1.9}$$

In words, AID provides an expression of the exact Jacobian $\boldsymbol{J}_{\boldsymbol{W}^\star}(\boldsymbol{\Theta})$, hence the hypergradient, when the matrix $\frac{\partial^2 F_{in}(\boldsymbol{W}^\star(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \boldsymbol{W}}$ is invertible. In practice, the hypergradient is not calculated using the resulting expression, rather it is approximated using numerical methods, similar to the ITD method. This is due to the following facts:

- $\boldsymbol{W}^\star$ is typically not available in a closed-form formula that can be evaluated, and is expensive to compute exactly using alternative iterative methods.
- inverting the matrix $\frac{\partial^2 F_{in}(\boldsymbol{W}^\star(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \boldsymbol{W}}$ is expensive especially when the number of inner variables is large.

To overcome the first issue, we replace $\boldsymbol{W}^\star$ by the output $\boldsymbol{W}_{\tau_{in}}$ obtained after τ_{in} iterations by the inner optimizer. Consequently, the first step in approximating the hypergradient consists in substituting $\boldsymbol{W}^\star$ by $\boldsymbol{W}_{\tau_{in}}$ in Eq. (1.9). The second step is to approximate the inverse $\Big(\frac{\partial^2 F_{in}(\boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \boldsymbol{W}}\Big)^{-1}$ appearing in the resulted expression. Many numerical methods can be used to accomplish that, among which there are two commonly used algorithms: *i)* solving a linear system (Pedregosa, 2016; Rajeswaran et al., 2019); *ii)* using truncated Neumann series (Lorraine et al., 2020). We present the former method for completeness.

Solving linear systems: this method directly approximates the inverse-Hessian-vector product $\Big(\frac{\partial^2 F_{in}(\boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \boldsymbol{W}}\Big)^{-1} \frac{\partial F_{out}\big(\boldsymbol{\Theta}, \boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta})\big)}{\partial \boldsymbol{W}}$ by solving the following linear system for $\boldsymbol{q}$:

$$\frac{\partial^2 F_{in}(\boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta}), \boldsymbol{\Theta})}{\partial \boldsymbol{W}^\top \boldsymbol{W}} \boldsymbol{q} = \frac{\partial F_{out}(\boldsymbol{\Theta}, \boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta}))}{\partial \boldsymbol{W}} \ ,$$

which yields lower computational and memory costs. This concludes the hypergradient approximation as other terms can be evaluated using AD.

One notices that AID, in contrast to ITD, is independent from the optimization trajectory leading from $\boldsymbol{W}_0$ to $\boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta})$. Therefore, AID is algorithm-agnostic with regards to the iterative approach that computes $\boldsymbol{W}_{\tau_{in}}(\boldsymbol{\Theta})$.

Remark 1.5.1 (ITD vs. AID)**.** When we started this work, there was not enough theoretical or empirical evidence on the superiority of one method over the other

(Grazzi et al., 2020). That is, AID was known to be less memory-demanding than ITD, but it was not clear whether it is more efficient approximating the hypergradient. Given that, and the fact that ITD is easier to implement thanks to machine learning packages, we decided to use it in approximating hypergradients. Meanwhile, recent studies started to slightly favor AID. With a set of assumptions including strong convexity and Lipschitz smoothness in the inner problem, Grazzi et al. (2020) showed that both methods have linear convergence rates with AID being slightly faster, and empirically validated their findings. Nonetheless, experiments showed that ITD is more reliable when these assumption are not met. Similarly and with different assumptions including F_{out}, its derivative, F_{in}, its derivative, and its second-order derivative being Lipschitz functions, and strong convexity in the inner problem, Ji (2021) derived sharper bounds and showed a slightly faster convergence rate for AID.

Remark 1.5.2 (Stochastic methods)**.** Likewise single-level problems, many methods tackled the bilevel optimization (1.7) with a stochastic version of ITD or AID, while motivated by the reduced cost thanks to the use of a single or a mini-batch of points to approximate terms rather than using the full dataset at each iteration. For instance, Maclaurin et al. (2015) make use of ITD when a Stochastic Gradient Descent (SGD) with momentum is employed in the inner problem, which proved efficient in hyperparameter optimization for machine learning tasks. Similarly, Ghadimi and Wang (2018); Ji (2021); Chen et al. (2021) proposed adopting SGD updates in the inner problem while also sampling mini-batches for the Neumann series-based AID to approximate hypergradients. Interestingly, Chen et al. (2021); Dagréou et al. (2022) show that their AID-based stochastic methods has a sample complexity that matches the one of SGD applied on single-level problems. As we will see in Chapter 2, we leverage the stochastic version of ITD in the first task we tackle in this thesis, since the dataset size in according experiments is large.

Remark 1.5.3 (The convexity assumption)**.** ITD and AID assume access to an iterative algorithm that converges to $\boldsymbol{W}^\star(\boldsymbol{\Theta})$. Furthermore, the use of a gradient-based algorithm on top of these methods to learn the outer variables implies that the bilevel problem is convex in $\boldsymbol{\Theta}$. We emphasize that this is not satisfied in this work, since the problems we consider are not convex in $\boldsymbol{\Theta}$, and only one of the three inner problem instances is convex in $\boldsymbol{W}$, specifically when the inner problem involves the Laplacian regularization model as presented in Section 1.6. In fact, this is usually the case in machine learning tasks, where in practice gradient-based algorithms are applied to obtain a good local minima instead. Despite the non-convex setting, we show empirically that ITD is still efficient when gradient-based optimizers are deployed in the inner and the outer problems, as it outputs graph structures that notably improve over the observed ones.

1.6 Graph learning with bilevel optimization

Having introduced bilevel optimization and the necessity of graph learning, we can now formulate the problem of graph learning using bilevel optimization. Formally, we consider the case where the graph objective function is a function of the *trained* graph-based model, which, in our work, can be a GNN or the Laplacian regularization model, chosen as a representative method for graph regularization and node embedding methodologies, respectively. Given a second set of labeled nodes $V_{out} \subset V$ distinct from V_{tr} used to train the graph-based model, the vector $\boldsymbol{Y}_{obs} \in \mathbb{R}^n$ which contains the labels of nodes at coordinates $i \in V_{tr} \cup V_{out}$ and, *e.g.,* not-a-number "*NaN*" outside of $V_{tr} \cup V_{out}$, and a set of admissible adjacency matrices $\mathcal{A}$, the bilevel graph optimization is cast as

$$\boldsymbol{A}^\star \in \arg\min_{\boldsymbol{A}\in\mathcal{A}} F_{out} = \frac{1}{|V_{out}|} \sum_{i\in V_{out}} \ell(\boldsymbol{Y}(\boldsymbol{A})_i, (\boldsymbol{Y}_{obs})_i), \tag{1.10}$$

such that $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{GNN}(\boldsymbol{X}, \boldsymbol{A})$ (GNN case) or $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A})$ (the Laplacian regularization case). That is, the graph learning problem, called the *outer* problem, is a constrained optimization, where its constraint involves the output of the *inner* optimization problem $\boldsymbol{Y}(\boldsymbol{A})$: either (1.2) over $\boldsymbol{Y}$ or (1.5) over W. One may add a regularization term to F_{out} to impose some regularity or priors on the generated graph. In fact, we do that in Section 1.7.3 and Chapter 4 in order to alleviate an issue in graph learning, which we analyse in the aforementioned chapter. If no regularization term is added, F_{out} can be seen as the validation loss in classification/regression tasks, and V_{out} can be seen as the validation set.

Several models are possible for $\mathcal{A}$ when directly optimizing for the graph including

- **Full learning**: $\mathcal{A} = [a, b]^{n\times n}$ is the set of all weighted adjacency matrices (generally with some bounds a, b on the weights). This choice necessarily leads to an impractical quadratic complexity on the minimization.
- **Edge refinement**: the learned adjacency matrix has the same zero-pattern as the observed adjacency matrix, that is, we learn weights only on existing edges.
  $$\mathcal{A} = \{\boldsymbol{A} \in [a, b]^{n\times n} | \boldsymbol{A}_{i,j} = 0 \text{ when } (\boldsymbol{A}_{obs})_{i,j} = 0\}.$$
  The complexity is proportional to the number of edges, generally less than quadratic in n as graphs tend to be sparse.
- **Generalized edge refinement**: same principle, but the zero-pattern is given by a modification of the observed adjacency matrix. For instance, taking the zero-pattern of $\boldsymbol{A}^r_{obs}$ yields an edge between neighbors that are less than r-hop from each other in $\mathcal{G}_{obs}$.

Another model for $\mathcal{A}$ can be obtained by adopting metric learning:

- **Metric Learning**: the learned graph is the output of a model f_Θ parameterized by the weights Θ. The model takes as input node features and the observed graph, that is, $\mathcal{A} = \{\boldsymbol{A}_\Theta = f_\Theta(\boldsymbol{A}_{obs}, \boldsymbol{X})\}$. In this scenario, the outer optimization problem is carried out on Θ instead of $\boldsymbol{A}$, *i.e.,* the bilevel optimization is cast as

$$\Theta^\star \in \arg\min_\Theta F_{out} = \frac{1}{|V_{out}|} \sum_{i \in V_{out}} \ell(\boldsymbol{Y}(\boldsymbol{A}_\Theta)_i, (\boldsymbol{Y}_{obs})_i), \tag{1.11}$$

  such that $\boldsymbol{Y}(\boldsymbol{A}_\Theta) = \boldsymbol{Y}_{GNN}(\boldsymbol{X}, \boldsymbol{A}_\Theta)$ or $\boldsymbol{Y}(\boldsymbol{A}_\Theta) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A}_\Theta)$. For the best of our knowledge, the metric learning model has not been considered in the literature with bilevel optimization. In fact, adopting this model is one of our contributions that we highlight in Section 1.7.2 and discuss in details in Chapter 3.

Note that in addition to SSL tasks that are of interest in this thesis, the bilevel framework can be applied to *any* supervised learning problem too.

1.7 Contribution

The contribution of this thesis is threefold. In the first part, and as a simple starting point, we look at the problem of learning analysis-sparsity priors with bilevel optimization, which will be introduced to the reader in the next section. The second part presents a novel framework for learning graph structures, which consists in adopting metric learning in the bilevel framework. That is, we train a parametric model on predicting edge weights. In the last part, we identify a lack of supervision induced when optimizing directly for the graph in the bilevel framework under the edge refinement setting. We give a precise mathematical characterization of this phenomenon for different graph-based models, and examine possible solutions to it, including the framework proposed in the second part.

1.7.1 Warming up: learning analysis-sparsity priors with bilevel optimization

Denoising is a widely-tackled problem that emerges in many fields, ranging from biomedical engineering (McCann et al., 2019) to computer vision (Yang et al., 2017) and remote sensing (Addesso et al., 2017). The goal is to restore the signal from its noisy observations. Usually, the model of the imaging system is *a priori* known: $\boldsymbol{y} = \boldsymbol{w} + \boldsymbol{\varepsilon}$, where $\boldsymbol{w}, \boldsymbol{y} \in \mathbb{R}^p$ are the true and the measured signals, respectively, and $\boldsymbol{\varepsilon} \in \mathbb{R}^p$ is additive noise. In addition, a prior hypothesis on the

nature of the signal might be available, like sparsity (McCann and Ravishankar, 2020). Such extra knowledge can be incorporated in the optimization process to get better reconstructions, *e.g.,* a higher Signal to Noise Ratio (SNR).

Sparsity priors exist in two forms (Elad et al., 2007): *i)* synthesis (traditional) sparsity where $\boldsymbol{w} = \boldsymbol{\Omega}\boldsymbol{u}$, $\boldsymbol{\Omega}$ is a linear operator, and $\boldsymbol{u}$ is sparse; *ii)* analysis-sparsity where $\boldsymbol{v} = \boldsymbol{\Omega}^\top \boldsymbol{w} \in \mathbb{R}^m$ is sparse. This section is interested in the latter. As a convex surrogate, this prior is enforced on $\boldsymbol{w}$ by adding the term $\|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1$ to the (generally quadratic) loss function (Mancera and Portilla, 2006), where $\|\cdot\|_1$ is the ℓ_1 norm. Putting all together, the problem consists in finding: $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1$ for some regularization amplitude parameter $\lambda \geq 0$. The linear operator $\boldsymbol{\Omega}$ is either user-defined, or learned directly from data.

The main problem we tackle in this part is to extract both $\boldsymbol{\Omega}$ and λ from data for denoising tasks with supervised learning. Therefore, the downstream task here is not a graph-based SSL problem. Having that $\lambda \|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1 = \|(\lambda \boldsymbol{\Omega})^\top \boldsymbol{w}\|_1$, this problem is equivalent to extracting the product $\lambda\boldsymbol{\Omega}$ as one object, thus we stop writing λ explicitly from now on and keep $\boldsymbol{\Omega}$. Formally, the task we are interested in is the following: having a dataset $(\boldsymbol{y}_l, \boldsymbol{w}_l)_{l=1}^L$ of L pairs of measurements and its associated ground-truth signals, find the operator $\boldsymbol{\Omega}$ that minimizes the mean squared error between reconstructions and ground truth signals:

$$\min_{\boldsymbol{\Omega} \in \mathcal{A}} \sum_{l=1}^{L} \left\| \hat{\boldsymbol{w}}\left(\boldsymbol{\Omega}, \boldsymbol{y}_l\right) - \boldsymbol{w}_l \right\|_2^2 \tag{1.12a}$$

$$s.t. \quad \hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y}) = \arg\min_{\boldsymbol{w} \in \mathbb{R}^p} \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{w}\|_2^2 + \|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1 \ . \tag{1.12b}$$

Equation (1.12) is a *bilevel optimization problem*: in the outer problem, we optimize for the dictionary $\boldsymbol{\Omega}$, while in the inner problem we denoise measurements. That is, $\Theta = \{\boldsymbol{\Omega}\}, W^\star(\Theta) = \{\hat{\boldsymbol{w}}\left(\boldsymbol{\Omega}, \boldsymbol{y}_l\right)\}_{l=1}^L$. We already know that the inner part can be solved applying the Forward-Backward splitting (FB) algorithm on the dual problem (Chambolle et al., 2010). However, due to the ℓ_1 norm, neither the solution nor its gradient *w.r.t.* $\boldsymbol{\Omega}$ have a closed-form expression. Thus, the solution of the bilevel problem cannot be derived analytically nor obtained with gradient-based methods.

Contribution: we approximately recover the analysis-sparsity operator employing ITerative Differentiation (ITD) by unrolling the FB algorithm applied on the dual problem of Eq. (1.12b). This is, for the best of our knowledge, the first work that uses ITD to learn analysis-sparsity priors as in Eq. (1.12), which is highly non-smooth, without relying on any relaxation technique. This permits to examine the

capacity of ITD, the AD phase in particular, in such setting. Indeed, experiments prove the effectiveness of ITD in learning the operator from 1D piecewise constant signals and from 2D images. Moreover, we propose to reduce the admissible set $\mathcal{A}$ to dictionaries with columns summing up to zero when learning from piecewise constant signals, and empirically prove that this increases stability and extracts an operator with higher quality than a previous baseline method.

Why this problem as a starting point?

As previously stated, this problem serves as a starting point before proceeding to the problem of graph learning. This is due to the following reasons:

- the FB updates on the inner level are available in a closed-form expression, thereby AD is required to perform first-order differentiation to compute the hypergradient. In contrast, the inner problem in the graph learning case (*e.g.,* learning the weights of a GNN model) is optimized with a gradient-based method with gradients being evaluated via AD. That is, AD for graph learning is required to evaluate gradients for the inner problem, and *gradients of gradients* to compute the hypergradient. Therefore, (1.12) is a good starting point to get familiar with the use of AD in the context of bilevel optimization.
- problem (1.12) is non-smooth which is a challenging setting for ITD. Given that Peyré and Fadili (2011) relaxed Eq. (1.12) and derived a formula of the hypergradient, we can validate the ITD's performance in this setting by comparing its output to the hypergradient of the relaxed problem.
- problem (1.12) is indeed used in the context of graph learning in many applications. In fact, when the coefficients in signals $\boldsymbol{w}_l$ lie on a graph and are known to be neighborhood-wise constant, which means $\{(\boldsymbol{w}_l)_i - (\boldsymbol{w}_l)_j\}_{(i,j)\in E}$ is sparse, (1.12) is used to learn the graph by learning its incidence matrix, *i.e.,* $\mathcal{A}$ is the set of all incidence matrices of m edges.

Related work: problem (1.12) was first posed in Peyré and Fadili (2011), where authors smoothed the ℓ_1 norm so that the hypergradient has a closed-form expression. Similarly in Sprechmann et al. (2013), a different smoothing regime is adopted. However, the sought for sparsity is degraded with similar regimes that smooth ℓ_1 at *zero* (Nikolova, 2000). Recently in McCann and Ravishankar (2020), a formula of the hypergradient has been derived under some conditions, but it includes iteratively inverting a large matrix for each data point, which makes its implementation impractical. Chambolle and Pock (2021) conduct sensitivity analysis to compute hypergradients to learn *convolution-type* dictionaries with small support for piecewise constant signals. Such strong constraints are not considered

in our work; however, we will see that simple column-centering suffices to learn a high-quality dictionary. Another class of methods deploy learned neural networks to perform regularization (Kobler et al., 2020; Lecouat et al., 2020). Although data-driven regularizers are argued to leverage large amounts of data, analysis-sparsity regularizers guarantee that $\boldsymbol{\Omega}^\top \hat{\boldsymbol{w}}$ is zero-valued on large zones that are insensitive to small perturbations of data (Nikolova, 2000). Finally, the major difference against aforementioned methods is the use of AD to get hypergradients in such *non-smooth setting*, without using smoothing techniques, nor analytically deriving an algorithm that outputs this gradient when it is defined. We prove with empirical results the capacity of this framework.

1.7.2 Learning graph-to-graph models with bilevel optimization

In the second part of this thesis, we look at the problem of real-world graphs being corrupted or not given, which degrades performance in graph-based Semi-Supervised Learning (SSL) tasks. We address this problem by *learning graphs* of high-quality. More specifically, we look at the case where the objective guiding the learning process is to improve the performance of the trained graph-based model in the downstream task. That is, we look at a *bilevel optimization*.

The most straightforward idea is to deploy the bilevel optimization (1.10) under the edge refinement setting or the full learning setting. For the former case, we show in Section 1.7.3 and Chapter 4 that this induces a phenomenon of lack of supervision that we refer to as *gradient scarcity*, which notably degrades the quality of the learned graph. The latter case, on the other hand, is not practical due to the quadratic complexity in the number of nodes n.

Instead, we propose to solve the bilevel optimization problem under the metric learning setting as in Eq. (1.11). In words, we train a *parametric model* to learn a pair-wise similarity metric between nodes. This model takes in input the features of a pair of nodes and the observed edge weight between them and outputs the optimized edge weight. We refer to this model by G2G (Graph to Graph), named with inspiration from Set2Graph models (Serviansky et al., 2020). Indeed when the observed graph is edgeless, G2G is a function from sets to graphs. More details on the G2G structure is available in Section 3.3.1.

The bilevel problem (1.11) is intractable as neither the solution of the inner problem nor its gradient *w.r.t.* Θ has a closed form expression that can be evaluated. Hence, $\Theta^\star$ cannot be evaluated nor computed iteratively by a gradient-based algorithm. In addition, and as it is usually the case in modern machine learning, the outer problem is non-convex, thus we don't adopt finding an optimizer $\Theta^\star$, but

rather a good set of weights that proves the efficiency of our algorithm compared to baselines operating on the observed graph.

Contribution: we propose to train the G2G model by solving Eq. (1.11) via ITD. Remark here that the AD evaluates *gradients of gradients* as it: *i)* computes $\nabla_{\boldsymbol{W}} F_{in}$ for the inner updates, *ii)* evaluates the gradient of the output after these updates *w.r.t.* Θ. We optimize Θ afterwards using a gradient-based algorithm. To our knowledge, this is the first work that trains a G2G model through a bilevel optimization framework. The resulted G2G model then can be employed to reconstruct a high-quality graph. Even though we focus on the transductive setting, it is noteworthy that this model, once trained, can still be employed when adding new points to the dataset. That is, unlike when optimizing directly for the graph, the G2G model generalizes to the inductive setting and can be used to construct graphs for new datasets. Experiments on SSL datasets prove that our framework considerably outperforms models operating on the observed graph.

The reader will see in the third part (Section 1.7.3 and Chapter 4) that, in addition to the previously stated advantages of using G2G over directly optimizing for the graph, G2G models alleviate the gradient scarcity issue emerging in edge refinement tasks on SSL datasets.

1.7.3 Gradient scarcity in graph learning with bilevel optimization

In this part we extensively study the problem of *gradient scarcity* which appears when directly optimizing edge weights in the observed graph for graph-based SSL tasks. Gradient scarcity refers to the fact that edges between unlabeled nodes "far" from the labeled ones receive *zero gradients*, *i.e.,* they receive no supervision during the optimization and are not learned.

Fatemi et al. (2021) observed gradient scarcity when learning the graph and a GNN model with joint optimization explained in Section 1.4.2. Indeed, a k-layer GNN computes the label of a node using information from nodes at most k-hop far from it. This label is then not a function of edges connecting nodes outside of this neighborhood, and the term in the labeling loss corresponding to this label returns null gradients on those distant edges. In words, gradient scarcity is due to the finite receptive field (depth) of message-passing GNNs.

However, it is not straightforward how to extend this argument to the bilevel optimization (1.10) under the edge refinement setting. Specifically, the previous discussion assumes that the trained weights of the GNN after undergoing gradient-based updates do not depend on the adjacency matrix $\boldsymbol{A}$, which is not the case

in bilevel optimization. Moreover, if the problem holds in the bilevel setting, the roles of V_{tr} and V_{out} need to be clarified. Another question is if this problem is mitigated by resorting to graph-based models with infinite receptive field, *e.g.,* the Laplacian regularization.

Contributions: we prove that hypergradient scarcity occurs under the bilevel optimization setting when adopting GNNs. We show that using a k-layer GNN induces null hypergradients on edges between nodes at least k-hop from $V_{tr} \cup V_{out}$. For the Laplacian regularization, we prove that the problem persists, as hypergradients are exponentially damped with distance from labeled nodes. We empirically validate our findings. Then, we test three possible strategies to solve this issue: metric learning with G2G models, graph regularization and generalized edge refinement. Furthermore, we empirically distinguish between hypergradient scarcity and overfitting, in the sense that solving the former does not necessarily resolve the latter. To the best of our knowledge, this is the first work that mathematically tackles the gradient scarcity problem for bilevel optimization of graphs, and examines the phenomenon for models with infinite receptive field.

The main theorems we prove are the following:

Hypergradient scarcity with GNNs: we consider the bilevel optimization (1.10) adopting the GNN method $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{GNN}(\boldsymbol{X}, \boldsymbol{A})$.

Theorem 1.7.1. *Let $\boldsymbol{Y}_W$ be a k-layer GNN parametrized by the set of weights W. Assume that the inner optimization problem is solved with a gradient-based algorithm. Then, for any pair of nodes i, j at least k-hop from nodes in $V_{out} \cup V_{tr}$, we have $\frac{\partial F_{out}}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$.*

Note that with GNNs, the inner optimization is not a convex problem thereby we optimize to obtain a good local minima. However, our analysis demonstrates that hypergradient scarcity occurs precisely in this scenario, and does not necessitate the gradient-based algorithm to converge to the inner problem's optimizer.

Hypergradient scarcity with the Laplacian regularization: We show that although to a lesser degree, this issue still arises when adopting the Laplacian regularization, *i.e.,* $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A})$. Specifically, we establish that the hypergradient's magnitude diminishes *exponentially* as the sum of the two distances to V_{tr} and V_{out} increases. Our study is focused on regression tasks, where ℓ is the MSE loss function in Eqs. (1.1) and (1.10). Let $\boldsymbol{S}_{in} \in \mathbb{R}^{n \times n}$ be the diagonal matrix with entries equal to 1 for nodes in V_{tr} and 0 otherwise, the solution $\boldsymbol{Y}(\boldsymbol{A})$ enjoys a closed-form expression:

$$\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{B}^{-1} \tilde{\boldsymbol{S}}_{in} \boldsymbol{Y}_{obs} \ ,$$

where $\tilde{\boldsymbol{S}}_{in} = \frac{\boldsymbol{S}_{in}}{|V_{tr}|}$ and $\boldsymbol{B} = \frac{\boldsymbol{S}_{in}}{|V_{tr}|} + \lambda \frac{\boldsymbol{L}}{|E|}$. Given that, we now state the main result for the Laplacian regularization scenario.

Theorem 1.7.2. *Let nodes i, j be at least k-hop from V_{out}, and q-hop from V_{tr}. Then we have:*

$$\left| \frac{\partial F_{out}}{\partial \boldsymbol{A}_{ij}} \right| \lesssim \lambda \frac{\sqrt{|V_{out}|} + \mu_{\min}\sqrt{|V_{tr}|}|V_{out}|}{\mu_{\min}^3 |V_{tr}||E|} y_\infty^2 (1-\mu)^{q+k} \quad ,$$

s.t. $\mu_{\min}$ ($\mu_{\max}$) is the smallest (largest) eigenvalue of $\boldsymbol{B}$ which we prove to satisfy $0 < \mu_{\min} < \mu_{\max}$, $\mu = \frac{\mu_{\min}}{\mu_{\max}}$, and $y_\infty = \|\boldsymbol{Y}_{obs}\|_\infty$.

Since $0 < 1 - \mu < 1$, Theorem 1.7.2 states that the hypergradient is exponentially damped as $q + k$ increases.

1.8 Outline

In Chapter 2, we employ bilevel optimization to learn analysis-sparsity operators for regularizing denoising tasks as in Eq. (1.12). We show how the inner optimization, *i.e.,* the denoising task, can be addressed with the Forward-Backward splitting (FB) algorithm. We then present our proposed method, which applies ITD to approximate hypergradients without relaxing non-smooth terms, and considers operators with columns summing up to zero when learning from piecewise constant signals (Algorithm 1). We empirically prove the effectiveness of this method in learning operators from 1D piecewise constant signals and 2D images.

Publications/preprints

Hashem Ghanem, Joseph Salmon, Nicolas Keriven, Samuel Vaiter. Supervised learning of analysis-sparsity priors with automatic differentiation. IEEE Signal Processing Letters, 2023.

We proceed to graph learning to improve performance in graph-based SSL tasks in Chapter 3. We show that this problem can be naturally cast as a bilevel optimization. We then propose to train a deep G2G model on predicting edge weights instead of directly optimizing for the graph. After proposing the model structure, we present the ITD-based bilevel optimizer (Algorithm 2), and emphasize that AD performs first order and second order differentiation in this task. Afterwards, we empirically demonstrate that our framework outputs graphs that notably improve the performance of graph-based methods compared to observed graphs.

Publications/preprints

Hashem Ghanem, Nicolas Keriven, Joseph Salmon, Samuel Vaiter. Supervised graph learning with bilevel optimization. 18th International Workshop on Mining and Learning with Graphs, 2022.

In Chapter 4, we study gradient scarcity which emerges when optimizing the weights of observed edges to minimize a labelling loss under the SSL setting, *i.e.,* label-based supervised graph learning as in Section 1.4.2. We first define this problem by edges between nodes far from the labelled set receiving null gradients during the optimization. Then, we prove that gradient scarcity emerges with bilevel optimization when adopting GNNs as the graph-based model as in Theorem 4.5.3. We next prove that it also emerges when adopting the Laplacian regularization model, and that hypergradients are exponentially damped with distance from labeled nodes (Theorem 4.6.2). We empirically validate our findings, then we show that among the following strategies: metric learning with G2G models, graph regularization and generalized edge refinement, the first two are efficient in resolving this issue.

Publications/preprints

Hashem Ghanem, Samuel Vaiter, Nicolas Keriven. Gradient scarcity with bilevel optimization for graph learning. In arXiv preprint, 2023.

Chapter 2

Bilevel learning of analysis-sparsity priors

In this chapter, we deploy ITerative Differentiation (ITD) to learn analysis-sparsity operators with bilevel optimization. We introduce the analysis-sparsity regularization, and formulate learning its operator from data as a bilevel optimization problem in Section 2.1. Next, in Section 2.2, we review related works which often smooth the inner problem, then we point out the importance of preserving the non-smoothness. Consequently in Section 2.3, we propose to apply ITD to learn the operator using projected gradient descent without smoothing the inner problem. In addition to preserving non-smoothness, this permits to examine the capacity of ITD in non-smooth settings. We also restrict the search to operators with 0-centered columns when learning from piecewise constant signals, or from signals which lie on a graph and are neighborhood-wise constant, and empirically show that this removes undesired local minima and improves numerical stability. In Section 2.4, we empirically show that the proposed method successfully recovers the analysis-sparsity operator from 1D piecewise constant signals and validate its efficiency in 2D image denoising.

This chapter presents the content of our publication Ghanem et al. (2023a).

2.1 Introduction

Denoising consists in restoring signals from noisy observations. Usually, the model of the imaging system is *a priori* known: $\boldsymbol{y} = \boldsymbol{w} + \boldsymbol{\varepsilon}$, where $\boldsymbol{w}, \boldsymbol{y} \in \mathbb{R}^p$ are the true and the measured signals, respectively, and $\boldsymbol{\varepsilon} \in \mathbb{R}^p$ is additive noise. In addition, a prior hypothesis on the nature of the signal might be available, like sparsity

(McCann and Ravishankar, 2020). Such extra knowledge can be incorporated in the optimization process to get better reconstructions (*e.g.,* a higher Signal to Noise Ratio (SNR)).

Sparsity priors exist in two forms (Elad et al., 2007): *i)* synthesis (traditional) sparsity where $\boldsymbol{w} = \boldsymbol{\Omega}\boldsymbol{u}$, $\boldsymbol{\Omega}$ is a linear operator, and $\boldsymbol{u}$ is sparse; *ii)* analysis-sparsity where $\boldsymbol{v} = \boldsymbol{\Omega}^\top \boldsymbol{w} \in \mathbb{R}^m$ is sparse. This work is interested in the latter. As a convex surrogate, this prior is enforced on $\boldsymbol{w}$ by adding the term $\|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1$ to the (generally quadratic) loss function (Mancera and Portilla, 2006), where $\|\cdot\|_1$ is the ℓ_1 norm. Putting all together, the problem consists in finding: $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1$ for some regularization amplitude $\lambda \geq 0$. The linear operator $\boldsymbol{\Omega}$ is either user-defined, or learned directly from data.

The main problem we tackle in this chapter is to extract both $\boldsymbol{\Omega}$ and λ from data with supervised learning. Having that $\lambda\|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1 = \|(\lambda\boldsymbol{\Omega})^\top \boldsymbol{w}\|_1$, this problem is equivalent to extracting the product $\lambda\boldsymbol{\Omega}$ as one object, thus we stop writing λ explicitly from now on and keep $\boldsymbol{\Omega}$. To avoid trivial solutions and undesired local minima, it is common to restrict the search space to an admissible set $\mathcal{A} \subset \mathbb{R}^{p\times m}$, where dictionaries have a specific property. In Ravishankar and Bresler (2015) for instance, $\boldsymbol{\Omega}$ is forced to be orthogonal, *i.e.,* $\mathcal{A} = \{\boldsymbol{\Omega}; \boldsymbol{\Omega}\boldsymbol{\Omega}^\top = \boldsymbol{I}_p\}$. In Peyré and Fadili (2011), $\boldsymbol{\Omega}$ is constrained to be a convolution dictionary. However, it is still challenging to find an admissible set that performs well in all applications of this problem (Yaghoobi et al., 2012). In this work, we do not commit to find such universal set. The task we are interested in is the following: having a dataset $(\boldsymbol{y}_l, \boldsymbol{w}_l)_{l=1}^L$ of L pairs of measurements and its associated ground-truth signals, find the operator $\boldsymbol{\Omega}$ that minimizes the mean squared error between reconstructions and ground truth signals:

$$\boldsymbol{\Omega}^\star \in \arg\min_{\boldsymbol{\Omega}\in\mathcal{A}} F_{out} = \sum_{l=1}^{L} \left\|\hat{\boldsymbol{w}}\left(\boldsymbol{\Omega}, \boldsymbol{y}_l\right) - \boldsymbol{w}_l\right\|_2^2 \tag{2.1a}$$

$$s.t. \quad \hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y}) = \arg\min_{\boldsymbol{w}\in\mathbb{R}^p} F_{in} = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{w}\|_2^2 + \|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1 \ . \tag{2.1b}$$

Equation (2.1) is a *bilevel optimization problem* as in (1.7): in the outer problem, we optimize $\Theta = \{\boldsymbol{\Omega}\}$ as in Eq. (2.1a), while in the inner problem we denoise measurements to obtain $W^\star(\Theta) = \{\hat{\boldsymbol{w}}\left(\boldsymbol{\Omega}, \boldsymbol{y}_l\right)\}_{l=1}^L$ following Eq. (2.1b).

We already know that the inner part can be solved applying the Forward-Backward splitting (FB) algorithm on its dual problem (Chambolle et al., 2010), see Section 2.3.1. However, due to the ℓ_1 norm, neither the solution nor its gradient *w.r.t.* $\boldsymbol{\Omega}$ have closed form expressions. Thus, the minimizer $\boldsymbol{\Omega}^\star$ cannot be derived analytically nor obtained with gradient-based methods.

Before stating our contribution in mitigating this issue, we first present the two applications we consider to validate the efficiency of our method. The first one is the well-known problem of *1D piecewise constant signals reconstruction* (Chambolle et al., 2010), where given a signal $\boldsymbol{w} = (w_1 \dots, w_p)^\top$, this prior indicates that $(w_2 - w_1, w_3 - w_2, \dots, w_1 - w_p)^\top$ is sparse. The estimator is often written as an instance of Eq. (2.1b), with $\boldsymbol{\Omega} = \boldsymbol{\Omega}_{TV}$ the dictionary associated with the 1D Total Variation (TV) regularization: for all $i \in \{1, \dots, p\}; \boldsymbol{\Omega}_{i,i} = -1, \boldsymbol{\Omega}_{i+1,i} = 1$, and 0 otherwise, up to rescaling. The second application is 2D image denoising, where we deploy the learned dictionary to denoise image rows one by one to get one image, repeat the process column-wise to get another one, and finally average both results.

In this work, we propose to approximately recover the analysis-sparsity operator $\boldsymbol{\Omega}^\star$ by: *i)* replacing the true minimizer $\hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y})$ by the output of the FB algorithm applied on the dual problem of Eq. (2.1b); *ii)* deploying *Automatic Differentiation (AD)* to solve Eq. (2.1a) with projected gradient descent. That is the ITD algorithm detailed in Section 1.5.4. Although ITD is a standard approach to solve bilevel problems, this is, to the best of our knowledge, the first work that uses this method to learn analysis-sparsity priors as in Eq. (2.1), which is highly non-smooth, without relying on any relaxation technique. This permits to examine the capacity of AD in such setting. We empirically demonstrate that our method recovers the TV dictionary $\boldsymbol{\Omega}_{TV}$ from piecewise constant signals. Moreover, we reduce the admissible set $\mathcal{A}$ to dictionaries with columns summing up to zero for this experiment, and empirically prove that this increases stability and extracts the TV operator with higher quality than previous methods. We also show that our algorithm performs reasonably well on 2D image denoising, where we average row-wise and column-wise denoising.

2.2 Related work

The bilevel problem (2.1) was first posed in Peyré and Fadili (2011), where authors smoothed the ℓ_1 norm so that the derivative of $\hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y})$ *w.r.t.* $\boldsymbol{\Omega}$ has a closed form expression. Then, they applied gradient descent to find a local minimizer $\boldsymbol{\Omega}$. Similarly in Sprechmann et al. (2013), a different relaxation regime by means of the smooth ℓ_2 norm is adopted. However, the sought-for sparsity is degraded in the same manner than smooth ℓ_1 at *zero* (Nikolova, 2000). Recently in McCann and Ravishankar (2020), a formula of the gradient of the outer problem has been derived under some conditions, but it includes iteratively inverting a large matrix (with worst-case complexity in $\mathbb{R}^{(p+m)^2}$) for each data point. Even though it was shown to perform well on reasonably small datasets, this method is too expensive

for large datasets. The work in Chambolle and Pock (2021) solves the 2D piecewise signals reconstruction problem, and conducts sensitivity analysis to compute hypergradients to learn *convolution-type* dictionaries with small support. Such strong constraints are not considered in our work, however we will see that simple column-centering suffices to learn a high-quality dictionary in the 1D version of this problem.

Instead of considering off-the-shelf regularizers, a class of methods deploy learned neural networks to perform regularization. In Heaton et al. (2021); Gilton et al. (2021), the proposed models are trained by looking at inner updates as fixed-point iterations, which is leveraged to compute hypergradients instead of using ITD. These hypergradients are evaluated iteratively either with fixed-point updates or using Jacobian-free backpropagation (Fung et al., 2021). Fixed-point based optimization yields benefits *w.r.t.* memory, computation cost and accuracy, but, unlike ITD, it is not applicable on dynamic inner updates. In Kobler et al. (2020), a U-Net based regularizer is trained by writing the bilevel problem as an optimal control problem, where gradients are computed deriving adjoint state equations. This model achieved state-of-the-art results for image denoising with a low number of trained parameters. In Lecouat et al. (2020), the regularizer is a combination of classical priors, including total variation. The inner problem is iteratively solved within the context of non-cooperative games, then AD is used to get hypergradients. However, non-smooth terms are relaxed with Moreau-Yosida smoothing (Hiriart-Urruty and Lemaréchal, 2013). Still, the regularizer is interpretable and achieves competitive results on image processing applications. Although data-driven regularizers are argued to leverage large amounts of data, analysis-sparsity regularizers guarantee that $\boldsymbol{\Omega}^\top \hat{\boldsymbol{w}}$ is zero-valued on large zones that are insensitive to small perturbations of data (Nikolova, 2000). Thus, it leverages recovering signals *a priori* known to exhibit analysis-sparsity.

The major difference against the aforementioned methods is the use of AD to get hypergradients in such *non-smooth settings*, without using smoothing techniques, or analytically deriving an algorithm that outputs the hypergradient when it is defined. We prove with empirical results the capacity of this framework.

2.3 Proposed algorithm

We solve the *dual* problem of Eq. (2.1b) with Forward-Backward splitting (FB). Using Automatic Differentiation, we obtain gradients of the FB algorithm *w.r.t.* $\boldsymbol{\Omega}$. Hypergradients are then used to learn a local minimum using gradient descent, while projecting $\boldsymbol{\Omega}$ on the admissible set $\mathcal{A}$ at every iteration.

2.3.1 Deriving and solving the dual problem of Eq. (2.1b)

The term $\|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1$ in Eq. (2.1b) is neither differentiable *w.r.t.* $\boldsymbol{w}$, nor does it have a simple proximal operator that can be efficiently computed; see Appendix B.2. Hence, we cannot apply gradient descent or the FB algorithm directly to evaluate $\hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y})$. However, the latter is possible if we tackle this optimization from the dual perspective (Rockafellar, 1974). In fact, one can prove that Eq. (2.1b) is equivalent to its dual problem (Chambolle et al., 2010):

$$\hat{\boldsymbol{z}}(\boldsymbol{\Omega}, \boldsymbol{y}) = \underset{\boldsymbol{z} \in \mathbb{R}^m}{\arg\min} \ \frac{1}{2} \big\| \boldsymbol{\Omega} \boldsymbol{z} - \boldsymbol{y} \big\|_2^2 \ + \ \iota_{B_\infty}(\boldsymbol{z}) \ , \tag{2.2}$$

where $B_\infty = \{\boldsymbol{z} \in \mathbb{R}^m; \ |z_i| \leq 1, \ \forall i \in [m]\}$ is the unit ball of the ℓ_∞ norm. The target recovery $\hat{\boldsymbol{w}}$ is then given by:

$$\hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y}) = \boldsymbol{y} - \boldsymbol{\Omega} \ \hat{\boldsymbol{z}}(\boldsymbol{\Omega}, \boldsymbol{y}). \tag{2.3}$$

The dual problem in Eq. (2.2) can be solved with any FB algorithm. Here, we adopt the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) (Beck and Teboulle, 2009); see Appendix B.1.

2.3.2 Outer loop design to learn the dictionary

Let us first notice that when the admissible set $\mathcal{A} = \mathbb{R}^{p \times m}$, the solution $\boldsymbol{\Omega}^\star$ is *not unique* as $\|\cdot\|_1$ is invariant to the coefficients order in a vector, *e.g.,* $\|(u_1, u_2)^\top\|_1 = \|(u_2, u_1)^\top\|_1$. Thus, permuting columns in $\boldsymbol{\Omega}$ will lead to the same $\hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y})$ in Eq. (2.1b), which means the same cost $F_{out}(\boldsymbol{\Omega})$. Given this, further simple analysis can show that the problem is *not convex*. As we will see later in this section, we optimize the operator $\boldsymbol{\Omega}$ using a gradient-based algorithm, therefore we consider the "good" local minimum that our framework converges to. We also assume that the second dimension of $\boldsymbol{\Omega}$ is given, while optimizing for it might be the subject of a future work.

Towards our goal, we use AD to get hypergradients of F_{out}, by tracing the FB algorithm that solves Eq. (2.1b). In fact, to reduce the computation cost, we do not compute the full MSE but sample a batch of training signals at each iteration (see Alg. 1). Since proving the convergence of AD's Jacobian to the variational one is complicated in such non-smooth setting, and that computing $\hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y})$ with high precision is computationally expensive, we replace the true reconstruction $\hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y})$ with the output of the FB algorithm after τ_{in} iterations $\hat{\boldsymbol{w}}_{\tau_{in}}(\boldsymbol{\Omega}, \boldsymbol{y})$, and empirically show that it is a good proxy. In other words, we use ITD to obtain hypergradients. We randomly initialize $\boldsymbol{\Omega}_0$, then we start each iteration t by setting *PyTorch AD framework* to record operations on $\boldsymbol{\Omega}_t$. We can then compute

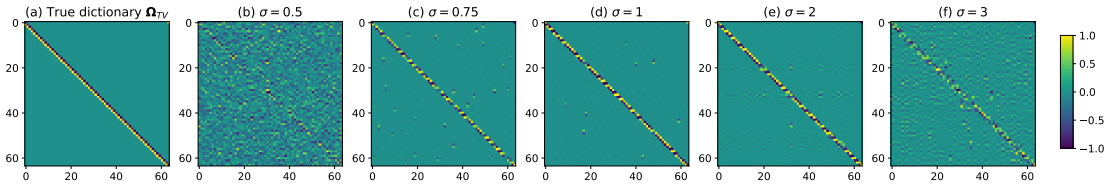


Figure 2.1 – Performance of our projected gradient descent algorithm 1, *w.r.t.* to noise level. We plot a sorted view of the dictionary $\mathbf{\Omega}^\star$. We rescale all dictionaries to $[-1, 1]$ for a better visualization of the structure recovered in $\mathbf{\Omega}^\star$. (a) $\mathbf{\Omega}_{TV}$ our algorithm is expected to learn. From (b) to (f): $\mathbf{\Omega}^\star$ for different values of σ, the standard deviation of noise. Jumps in the dataset have the same amplitude: $0 \to 10$ or $10 \to 0$.

the output $\hat{\boldsymbol{w}}_{\tau_{in}}(\mathbf{\Omega}, \boldsymbol{y})$ thereby $F_{out}(\mathbf{\Omega}_t)$. Now, we use the PyTorch AD to get the gradient $\nabla F_{out}(\mathbf{\Omega}_t)$, and update $\mathbf{\Omega}_t$ as follows:

$$\mathbf{\Omega}_{t+1} = \mathbf{\Omega}_t - \eta_{out} \nabla F_{out}(\mathbf{\Omega}_t) \ , \tag{2.4}$$

where $\eta_{out} > 0$ is a step size. Lastly, we project $\mathbf{\Omega}_t$ on the admissible set $\mathcal{A}$ by computing $\Pi_{\mathcal{A}}(\mathbf{\Omega}_t)$. We keep using the same notation $\mathbf{\Omega}^\star$ for the learned operator. The resulting algorithm is summarized in Algorithm 1.

2.4 Experiments

We conduct two sets of experiments, the first is on 1D synthetic signals and the second is on 2D real-world images.

Synthetic dataset: we consider the 1D piecewise constant signals reconstruction problem, with $p = m = 64$. Ground-truth examples $\boldsymbol{w}$ are generated s.t. they have 4 discontinuities. The coefficients where discontinuities take place are randomly chosen in each $\boldsymbol{w}$. Their amplitude varies through experiments and is specified whenever necessary. Observations $\boldsymbol{y}$ are constructed by adding a noise vector to each ground-truth signal, such vector is sampled from $\mathcal{N}(0, \sigma^2 \boldsymbol{I}_p)$, where σ varies through experiments. The "true" underlying dictionary $\mathbf{\Omega}_{TV}$ is shown in Fig. 2.1 (a), up to permuting its columns, and to rescaling with λ, which we compute in each experiment with a grid search solving Eq. (2.1a). Matrices $\mathbf{\Omega}^\star$ shown in this section have their columns sorted by magnitudes, to ease the comparison with $\mathbf{\Omega}_{TV}$. Training set size: 640000, validation set size: 256.

Real-world dataset: we use the MNIST dataset (LeCun, 1998), which includes images of handwritten digits. Therefore, images tend to be piecewise constant. We perform 2D image denoising by applying the 1D denoiser in Eq. (2.1b) on rows as

Algorithm 1: ITD-based projected gradient descent

Input: $\{(\boldsymbol{w}_l, \boldsymbol{y}_l)\}_{l\in\{1,\dots,L\}}$: dataset.
Output: $\boldsymbol{\Omega}^\star$: approximating a minimizer of $F_{out}(\boldsymbol{\Omega})$ in Eq. (2.1a).
Params: $m, \eta_{in}, \eta_{out}, \tau_{in}, \tau_{out}, batch\_sz$
Algorithm:
Initialize $\boldsymbol{\Omega}$ *iid* from $\mathcal{N}(0, 10^{-4})$.
Set PyTorch AD to track computations on $\boldsymbol{\Omega}$.
for $t \in \{0, \dots, \tau_{out} - 1\}$*:*
do
  $F_{out}(\boldsymbol{\Omega}) \leftarrow 0$
  for $l \in \{t * batch\_sz, \dots, (t+1) * batch\_sz - 1\}$
  do
    $\boldsymbol{z} \leftarrow \tau_{in}$ iterations of FISTA$(\boldsymbol{\Omega}, y_l, \eta_{in})$ as in Appendix B.1.
    $\hat{\boldsymbol{w}}_{\tau_{in}}(\boldsymbol{\Omega}, \boldsymbol{y}_l) \leftarrow \boldsymbol{y}_l - \boldsymbol{\Omega}\boldsymbol{z}$.
    $F_{out}(\boldsymbol{\Omega}) \leftarrow F_{out}(\boldsymbol{\Omega}) + \left\|\hat{\boldsymbol{w}}_{\tau_{in}}(\boldsymbol{\Omega}, \boldsymbol{y}_l) - \boldsymbol{w}_l\right\|_2^2$.
  $\nabla F_{out}(\boldsymbol{\Omega}) \leftarrow$ PyTorch AD gradient.
  $\boldsymbol{\Omega} \leftarrow \Pi_{\mathcal{A}}\big(\boldsymbol{\Omega} - \frac{\eta_{out}}{batch\_sz}\nabla F_{out}(\boldsymbol{\Omega})\big)$.
Return $\boldsymbol{\Omega}^\star = \boldsymbol{\Omega}$

well as on columns, then average the two resulted images. To this end, we extract all rows and columns in 28×28 MNIST images and contaminate these signals with *iid* additive Gaussian noise. The added noise is 0-mean and its standard deviation equals 0.04. Then, we use the output dataset to learn the dictionary $\boldsymbol{\Omega}^\star$ as in Eq. (2.1). We respect the standard training/validation split of MNIST dataset. Once training is performed, we deploy the trained denoiser for 2D denoising. We do not claim competing against state-of-the-arts on 2D image denoising, but we prove the capacity of our framework by showing its reasonably good performance on this task, evaluated by means of the SNR criterion.

Training setup: FISTA's step size η_{in}, refer to Appendix B.1, is assigned automatically to $\eta_{in} = 0.95/\|\boldsymbol{\Omega}^\top\boldsymbol{\Omega}\|_2$ to guarantee convergence in $\boldsymbol{z}$, where $\|\boldsymbol{\Omega}^\top\boldsymbol{\Omega}\|_2$ is the Lipschitz constant of $\nabla\frac{1}{2}\left\|\boldsymbol{\Omega}\boldsymbol{z} - \boldsymbol{y}\right\|_2^2$. We assume convergence in the inner problem, which determines the value of τ_{in}, if $\|\boldsymbol{z}_{i+1} - \boldsymbol{z}_i\|_\infty / \|\boldsymbol{z}_i\|_\infty < 10^{-4}$. We consider *stochastic gradient descent* updates with batch size 64 for the synthetic dataset, and 32 for MNIST. The step size η_{out} is set with grid search to 0.04 for synthetic data, and to 0.01 for MNIST. We adopt random white noise initialization with varying variance. We apply early stopping on the validation loss: $F_{out}(\boldsymbol{\Omega})$ evaluated on the validation set.

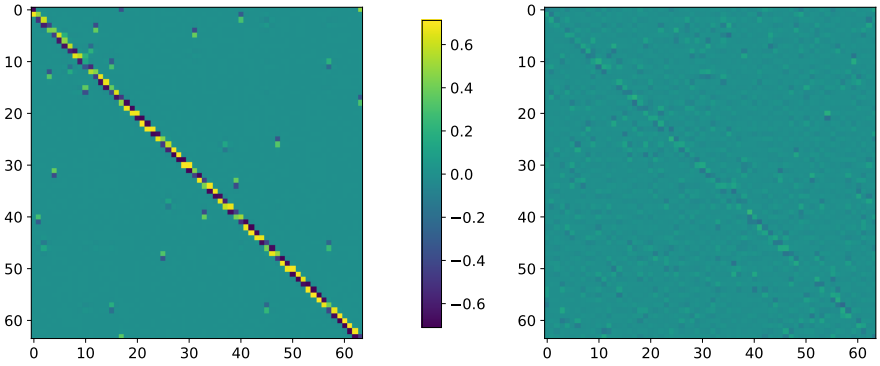


Figure 2.2 – The difference made by our proposed admissible set $\mathcal{A}$. We plot $\boldsymbol{\Omega}^\star$ produced by our algorithm. Left: with our proposed $\mathcal{A}$. Right: with $\mathcal{A} = \mathbb{R}^{p\times p}$. Jumps in the dataset can randomly occur between any two values in $[0, 10]$, thus they have random amplitudes. $\sigma = 0.5$.

2.4.1 Projection proposed for piecewise constant signals

We reduce the admissible set $\mathcal{A}$ in our proposed algorithm 1 to dictionaries whose columns sum up to zero, *i.e.*, $\mathcal{A} = \{\boldsymbol{\Omega} | \mathbf{1}_p^\top \boldsymbol{\Omega} = \mathbf{0}_m\}$. This property is seen in the prior operator $\boldsymbol{\Omega}_{TV}$ our algorithm is expected to learn from piecewise constant signals, and in the more general family of problems known as graph total variation (Berger et al., 2017), *i.e.*, learning from neighborhood-wise constant graph signals. This very simple prior greatly improves the results by filtering out many local minima. Different priors for other cases will be the goal of future investigations. Referring to the c-th column of $\boldsymbol{\Omega}$ by $\boldsymbol{\Omega}_{:,c}$, and by $\text{mean}(\boldsymbol{\Omega}_{:,c})$ to the mean value of this column, we project $\boldsymbol{\Omega}_{t+1}$ as follows:

$$(\boldsymbol{\Omega}_{t+1})_{:,c} = (\boldsymbol{\Omega}_{t+1})_{:,c} - \text{mean}\left((\boldsymbol{\Omega}_{t+1})_{:,c}\right), \ \forall c \in [m]. \tag{2.5}$$

In Fig. 2.2 and Fig. 2.3(left), we run our algorithm twice on the synthetic dataset: *i)* with the projection; *ii)* without the projection; and plot the learned dictionary $\boldsymbol{\Omega}^\star$ in both cases. Discontinuities in the used dataset are of random magnitudes (between any two values in [0,10]), and the noise has a standard deviation $\sigma = 0.5$. Our algorithm coupled with the projection successfully captures $\boldsymbol{\Omega}_{TV}$-like structure from the dataset, unlike when the projection is not considered, which shows its capability in this problem setting.

2.4.2 Sensitivity *w.r.t.* to the noise level

In this experiment, we consider the synthetic dataset, where signals $\boldsymbol{w}$ take values in $\{0, 10\}$. When observations are noise-free, *i.e.*, $\sigma = 0$ and $\boldsymbol{y} = \boldsymbol{w}$, it is clear

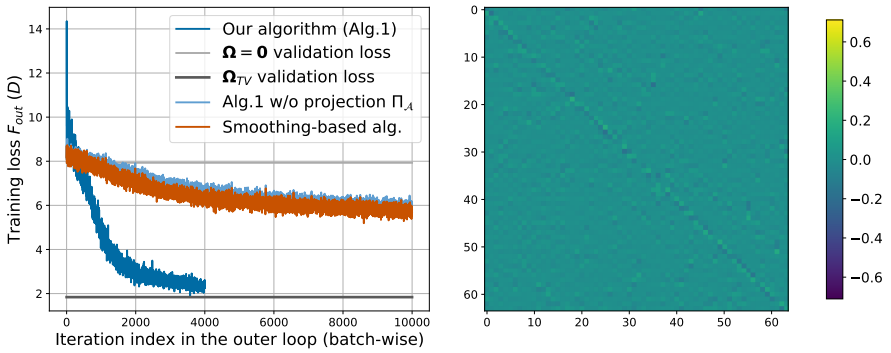


Figure 2.3 – Benchmark against the unconstrained optimization with ℓ_1-smoothing proposed in Peyré and Fadili (2011), with identical problem setting to Fig. 2.2. Left: training loss curves of different algorithms. Right: $\mathbf{\Omega}^\star$ produced by the algorithm in Peyré and Fadili (2011). Dataset: same as in Fig. 2.2.

$\mathbf{\Omega} = \mathbf{0}$ is the optimal dictionary, as no regularization is required to retrieve the true signal. Therefore, when σ is small, we don't expect our algorithm to learn the structure in $\mathbf{\Omega}_{TV}$, since in such case, its magnitude is of the same level as numerical errors. See Fig. 2.1 (b). On the other hand, when the noise level is high, the piecewise constant prior is degraded and poorly seen in observations. As a result, the learnt dictionary is a distorted version of $\mathbf{\Omega}_{TV}$, as in Fig. 2.1 (f).

In between, it is important to verify that our algorithm is stable, and can extract $\mathbf{\Omega}_{TV}$ out from data. As shown in Fig. 2.1 (c-e), this is indeed true when $\sigma \in [0.75, 2]$, which spans a non-trivial range in SNR scale $[12, 89]$. For $\sigma = 1$, we depict in Fig. 2.4 the evolution of losses through training, and the denoising performance on an observation from the validation set.

2.4.3 Benchmark against the algorithm proposed in Peyré et Fadili (2011)

In Fig. 2.3, we compare our output to the one of the unconstrained learning algorithm based on smoothing ℓ_1 in Peyré and Fadili (2011). We fix the ℓ_1-smoothing parameter $\epsilon = 10^{-3}$. The synthetic dataset here has discontinuities of random magnitudes in [0,10], and noise's standard deviation $\sigma = 0.5$, *i.e.,* same setup as in Fig. 2.2. Unlike our method, the other algorithm fails to learn the $\mathbf{\Omega}_{TV}$ structure from the data. Although the smoothing regime degrades reconstruction in the denoising phase, yet another main reason behind this failure is the absence of a projection like the centering projection, which helps our framework filtering out undesired local minima, like the one found by the algorithm in Peyré and Fadili (2011).

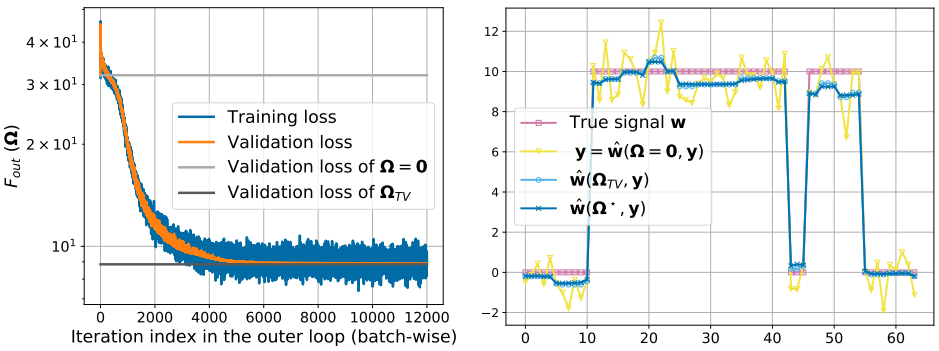
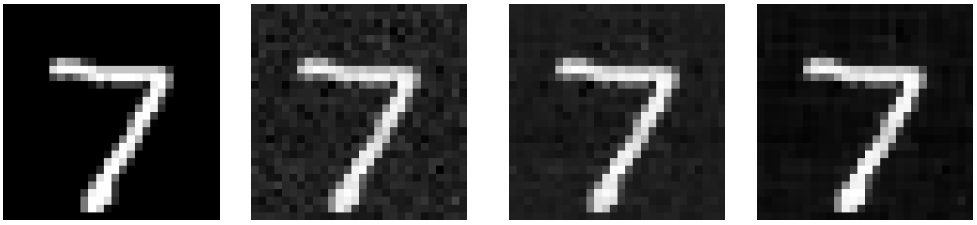


Figure 2.4 – Performance curves of the learnt $\mathbf{\Omega}^\star$ in the case $\sigma = 1$ in Fig. 2.1. Left: training and validation losses as in Eq. (2.1a) as a function of the iteration index. We plot the loss incurred by $\mathbf{\Omega}_{TV}$ and $\mathbf{\Omega} = \mathbf{0}$ on the validation set. Right: Recovered signals with $\mathbf{\Omega}_{TV}$, $\mathbf{\Omega} = \mathbf{0}$ and $\mathbf{\Omega}^\star$ as in Eq. (2.1b) from a random observation $\boldsymbol{y}$ in the validation set.



(a) Ground-truth $SNR = \infty$ (b) Observation $SNR = 14.92\ dB$ (c) Denoise w/$\mathbf{\Omega}^\star$ $SNR = 18.84\ dB$ (d) Denoise w/$\mathbf{\Omega}_{TV}$ $SNR = 19.95\ dB$

Figure 2.5 – Denoising performance on an MNIST image from the validation set. We report the SNR ratio in sub-captions.

2.4.4 2D image denoising on MNIST

Fig. 2.5 shows the denoising performance on an arbitrary image from the validation set. The learned dictionary $\mathbf{\Omega}^\star$ achieves a competitive SNR compared to $\mathbf{\Omega}_{TV}$, which is a popular choice for image denoising tasks, especially on piecewise constant images as in MNIST. Qualitatively, our framework struggles in learning the piecewise constant prior on pixels close to the image center, but succeeds on borders-nearby pixels. This is due to the fact that written digits are centered in most images in MNIST dataset, thus pixels close to the borders are more likely to equal zero, unlike pixels in the center whose value varies across images in the dataset.

Chapter 3

Bilevel learning of G2G models

From this point, we focus on graph learning with bilevel optimization. The simplest way to do that when an observed graph is given is to directly optimize the edge weights. Surprisingly, we show later in Chapter 4 that this leads to a phenomenon of lack of supervision that we term *gradient scarcity*, which significantly reduces the quality of the learned graph. Here in this chapter, we propose a novel framework to train G2G (Graph to Graph) models on predicting edge weights, which can be applied not only for edge refinement, but also for graph construction. In Section 3.1, we briefly restate the importance of graphs, and the motivation for graph learning. Then in the context of SSL, we formulate the problem as a bilevel optimization. Afterwards in Section 3.2, we review previous works that use observed labels for graph learning. In Section 3.3, we propose a framework that employs bilevel optimization to train parametric G2G models to learn a similarity metric between nodes. After presenting the proposed structure of G2G models, we then make use of ITD to solve the bilevel problem. Finally, in Section 3.5, we evaluate the proposed framework on SSL benchmark datasets and show that it notably improves performance compared to real-world observed graphs.

This chapter is based on the content of our workshop paper Ghanem et al. (2022).

3.1 Motivation and problem formulation

Graph-based learning has received increasing attention since data lying on graph structures is ubiquitous in many fields, ranging from social networks (Liben-Nowell and Kleinberg, 2003) to knowledge base (Ji et al., 2021), chemistry and medicine (Rong et al., 2020; Liu et al., 2018) and traffic (Wu et al., 2019). In these domains, it is important to deploy graph structures together with the given features to ex-

tract the sought for information. In social networks for example, the thoughts and beliefs of an individual are likely to be shared by his/her community and affected by what is called belief propagation through the network. Thus, leveraging the graph network is vital when solving member-level or network-level problems.

However, graph-based methods are sensitive to the graph quality, and unfortunately, graphs in practice are often noisy or not given. Usually in the former case the given graph is considered ground-truth, while in the latter case samples are processed as if they were mutually independent. This degrades the performance and leads to a sub-optimal solution.

In this work, we alleviate this issue by solving a *bilevel optimization problem* to train a model on capturing pairwise similarity between nodes, which eventually reflects the according edge weights. This model, referred to as G2G (Graph to Graph), takes in input node features and the observed graph. The name is inspired by *Set2Graph* models (Serviansky et al., 2020). Indeed, when the graph is edgeless, G2G is a function from sets to graphs. While this framework is capable of improving performance in *any* supervised or semi-supervised learning problem, we choose to specialize in the range of transductive SSL problems. We next present the graph-based models we consider to examine the efficiency of our framework, then we formulate the bilevel graph learning problem.

As reviewed in Section 1.3, there are roughly two main strategies to solve graph SSL problems. The first is to propagate known labels using a regularization process, and the second is to resort to node embedding methods. As a representative method for both strategies, we consider the Laplacian regularization and GNNs, respectively. Using Laplacian regularization, predicted labels read:

$$\boldsymbol{Y}_{\mathrm{Reg}}(\boldsymbol{A}) \in \underset{\boldsymbol{Y}\in\mathcal{B}}{\arg\min} \tfrac{1}{|V_{tr}|}\sum_{i\in V_{tr}} \ell(\boldsymbol{Y}_i, (\boldsymbol{Y}_{obs})_i) + \lambda \frac{1}{|E|}\sum_{i,j} \boldsymbol{A}_{i,j}\|\boldsymbol{Y}_i - \boldsymbol{Y}_j\|_2^2, \tag{3.1}$$

whereas for GNNs, the objective reads:

$$\begin{aligned} &\boldsymbol{Y}_{GNN} = \boldsymbol{Y}_{W^\star}(\boldsymbol{X}, \boldsymbol{A}), \text{ where} \\ &W^\star \in \underset{W}{\arg\min} \frac{1}{|V_{tr}|}\sum_{i\in V_{tr}} \ell\Big(\big(\boldsymbol{Y}_W(\boldsymbol{X}, \boldsymbol{A})\big)_i, (\boldsymbol{Y}_{obs})_i\Big) \ . \end{aligned} \tag{3.2}$$

Recall that the admissible set $\mathcal{B}$ is a subset of $\mathbb{R}^n$ in regression tasks, and $\mathbb{R}^{n\times C}$ in classification tasks where C is the number of classes, ℓ is a smooth loss function commonly chosen to be the Categorical Cross Entropy (CCE) loss for classification, and the Mean Square Error (MSE) for regression, λ is a balancing parameter, V_{tr} is the training set of labeled nodes, and W is the set of the GNN weights. The reader can refer to Section 1.3.2 regarding the GNN model we adopt in this work.

To train the G2G model, we consider the case where its objective is a function of the trained graph-based model, thereby, we look at a *bilevel optimization*. Denoting by Θ the weights of the G2G model, $\boldsymbol{A}_\Theta = G2G(\boldsymbol{X}, \boldsymbol{A}_{obs})$ its output adjacency matrix, and using a second set of labeled nodes $V_{out} \subset V$ distinct from V_{tr}, the bilevel optimization is cast as

$$\Theta^\star \in \arg\min_{\Theta} F_{out} = \frac{1}{|V_{out}|} \sum_{i \in V_{out}} \ell(\boldsymbol{Y}(\boldsymbol{A}_\Theta)_i, (\boldsymbol{Y}_{obs})_i), \tag{3.3}$$

such that $\boldsymbol{Y}(\boldsymbol{A}_\Theta) = \boldsymbol{Y}_{GNN}(\boldsymbol{X}, \boldsymbol{A}_\Theta)$ (GNN case) or $\boldsymbol{Y}(\boldsymbol{A}_\Theta) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A}_\Theta)$ (Laplacian regularization case). That is, the minimization of the outer objective function F_{out} involves $\boldsymbol{Y}(\boldsymbol{A}_\Theta)$, which is itself the result of an inner optimization problem, either (3.1) over $\boldsymbol{Y}$ or (3.2) over W. One may add a regularization term to F_{out} to impose some regularity or priors on the generated graph, but this is not considered in this chapter.

The bilevel problem (3.3) is intractable as neither the solution of the inner problem nor its gradient *w.r.t.* Θ has a closed form expression. Hence, $\Theta^\star$ cannot be evaluated nor computed iteratively by a gradient-based algorithm. In addition, and as it is usually the case in machine learning, this problem is non-convex, thus we don't adopt finding an optimizer $\Theta^\star$, but rather a good set of weights that proves the efficiency of our algorithm compared to baselines operating on the observed graph.

In the present work, we propose to train the G2G model with a gradient-based algorithm while using ITD to approximate hypergradients. This includes replacing the inner problem by a repeated application of any iterative algorithm that is guaranteed to converge to a good proxy. Here we consider gradient-based optimizers for the inner problem too. Then, we use automatic differentiation, more precisely higher-order automatic differentiation through the `Higher` package (Grefenstette et al., 2019), to trace these dynamics and finally compute $\nabla_{\boldsymbol{\Theta}} F_{out}$, where $\boldsymbol{\Theta}$ is the vectorized version of Θ. Similarly, we denote by $\boldsymbol{W}$ the vectorized version of W. Remark here that the `Higher` package evaluates *gradients of gradients* as it: *i)* computes $\nabla_{\boldsymbol{W}} F_{in}$ for the inner updates; *ii)* evaluates the gradient of the output after these updates *w.r.t.* Θ. The resulted G2G model can be then employed to reconstruct the underlying graph, even when adding new points to the dataset. In addition, we show that it is sufficient to trace the last few inner updates, here 10 iterations, to get a good approximation of $\nabla_{\boldsymbol{\Theta}} F_{out}$, which significantly reduce memory and time costs. To our knowledge, this is the first work that trains a G2G model through a bilevel optimization framework. Experiments on SSL datasets prove that our framework considerably outperforms models operating on the observed graph.

3.2 Related work

Bilevel optimization is used in many applications like multi-task and meta learning (Bennett et al., 2006; Flamary et al., 2014; Franceschi et al., 2018). See Section 1.5 for applications in other fields. Graph structure learning, on the other hand, gained in importance since the success of GNNs in relational learning, stemming from the fact that real-world graphs usually have corrupted edges. In this section, we review relevant works to ours which incorporate given labels in learning the graph structure. The reader can refer to Section 1.4 for a comprehensive review of other graph learners.

Franceschi et al. (2019) use bilevel optimization to learn the parameters of Bernoulli probability distributions over independent random edges, where these parameters are optimized to minimize the GNN's validation loss. This method includes learning n^2 parameters which limits scalability. Moreover, it does not generalize to new pairs of points, as this requires re-running the optimization process to learn the according parameters. This is mitigated in our proposed method, as the G2G model can be dynamically applied on new points to expand the constructed graph, and the number of its parameters does not depend on the dataset size, but rather on the problem complexity. On the other hand, Wan and Kokel (2021) solve a graph sparsification problem to remove edges from the observed graph by means of minimizing the GNN's validation loss, while keeping the graph connected. However, resulted graphs do not necessarily outperform the observed graph. We believe that the *hypergradient scarcity problem* that we characterize in Chapter 4 is the reason behind this.

Another set of methods learn both the graph and the graph-based model under the joint optimization setting. One example is the approach proposed by Wang and Leskovec (2020), where the weights of observed edges and a GNN model are jointly optimized to minimize the training loss. However, due to the large number of parameters, this often leads to overfitting. To address this issue, the authors incorporate the Label Propagation model (LP) to regularize the graph. The proposed framework produces state-of-the-art results on SSL node classification tasks. Likewise, in the work of Fatemi et al. (2021), the graph is regularized by imposing the assumption that a good graph must also perform well in denoising node features. For that end, the authors introduce a regularization penality which assesses the denoising performance.

Another advanced category of models that utilizes joint optimization are attention mechanisms, where the edge weights are re-evaluated after each GNN layer based on the similarity between node representations in that layer. The similarity criterion can either be user-defined, such as the dot product (Luong et al., 2015;

Vaswani et al., 2017), or learned locally at each layer by a single-layer feed-forward network (Veličković et al., 2018), or a combination of both approaches (Kim and Oh, 2021). These mechanisms proved effective in edge refinement tasks. In contrast to our method, these mechanisms are trained and used within a GNN model in one optimization problem.

In Stretcu et al. (2019), a deep model called the Graph Agreement Model (GAM) is trained on predicting edge weights. Unlike previous methods, the graph objective does not depend on a graph-based model and is not optimized under the bilevel nor the joint setting. Instead, it learns graphs by penalizing the absence of an edge between nodes with the same label. Once the GAM model is trained, its output graph is used to train a GNN model. The GNN model is then used to improve the set of labeled nodes by considering its confident predictions as ground truth. This training process is repeated for k iterations, and is called co-training (Blum and Mitchell, 1998).

3.3 Proposed method

We first present the structure of the G2G model adopted in our framework. Then, we state the bilevel learning routine we propose to train this model, with `Higher` package as a key ingredient to compute the hypergradients of problem (3.3).

3.3.1 G2G model design

Our proposed model takes as input the features $\boldsymbol{X}_i, \boldsymbol{X}_j$ of any two nodes i, j, their edge weight in the observed graph $(\boldsymbol{A}_{obs})_{i,j}$, and outputs a scalar edge weight $(\boldsymbol{A}_\Theta)_{i,j}$. It can be expressed as a combination of three functions:

$$(\boldsymbol{A}_\Theta)_{i,j} = \gamma\Big(\beta\big(\alpha(\boldsymbol{X}_i), \alpha(\boldsymbol{X}_j)\big), (\boldsymbol{A}_{obs})_{i,j}\Big) \ , \tag{3.4}$$

where:

- the encoder $\alpha : \mathbb{R}^p \to \mathbb{R}^{p_\alpha}$ is a Multi-Layer Perceptron (MLP) that computes a new representation vector for a node in a new embedding space of dimension p_α. A MLP network is composed of several fully connected feedforward layers. That is, given the output features $\boldsymbol{X}^{[l]}$ of the l-th layer, the output of the next layer is computed as follows:

$$\boldsymbol{X}^{[l+1]} = \phi^{[l+1]}(\boldsymbol{X}^{[l]}\boldsymbol{W}_1^{[l+1]} + \mathbf{1}_n(\boldsymbol{b}^{[l+1]})^\top) \ , \tag{3.5}$$

  where $\boldsymbol{W}_1^{[l+1]} \in \mathbb{R}^{d_l \times d_{l+1}}, \boldsymbol{b}^{[l+1]} \in \mathbb{R}^{d_{l+1}}$ are learnable parameters, d_l is the output dimensionality of the (l)-th layer, and ϕ is the non-linear activation function.

- the aggregator $\beta : \mathbb{R}^{p_\alpha} \times \mathbb{R}^{p_\alpha} \to \mathbb{R}^{p_\alpha}$ takes the embeddings of a pair of nodes from the previous stage α, and merges them to have a single representation vector in output. Let $\tilde{\boldsymbol{X}}_i, \tilde{\boldsymbol{X}}_j$ be the embeddings of nodes i, j, the aggregator we consider is the function:

$$\beta : (\tilde{\boldsymbol{X}}_i, \tilde{\boldsymbol{X}}_j) \mapsto (\tilde{\boldsymbol{X}}_i - \tilde{\boldsymbol{X}}_j)^2, \tag{3.6}$$

  where $(\cdot)^2$ is the square function applied at each dimension. One notices that this function is invariant to the order of its inputs.

- the regressor $\gamma : \mathbb{R}^{p_\alpha} \times \mathbb{R} \to [0, 1]$ is an MLP with sigmoid as the output activation function. For two nodes, γ takes the according aggregator output, the observed edge weight if provided, and outputs the sought for edge weight.

The proposed G2G architecture can be easily shown to be *permutation-equivariant*, that is, permuting the input graph permutes the output graph in the same manner. In mathematical terms, given any permutation function $\rho : [n] \to [n]$, where $[n] = \{1, \dots, n\}$, and $\boldsymbol{P} \in \{0, 1\}^{n \times n}$ the according permutation matrix, then the following condition holds:

$$G2G(\boldsymbol{P}\boldsymbol{X}, \boldsymbol{P}\boldsymbol{A}_{obs}\boldsymbol{P}) = \boldsymbol{P}G2G(\boldsymbol{X}, \boldsymbol{A}_{obs})\boldsymbol{P} \ . \tag{3.7}$$

Permutation equivariance (or invariance) is a critical attribute to enforce in graph processing. Here, it guarantees that the structure of the output graph is independent to relabeling the points in the input dataset, and can generalize to new graphs.

3.3.2 Learning routine

Neither the inner minimizer nor its gradient as a function of the graph used in training have a closed form expression in general. This makes it impossible to evaluate G2G optimal weights $\Theta^\star$ or learn it with a gradient-based algorithm. We propose to deploy ITD to evaluate hypergradients and learn Θ using a gradient-based method. This means that we first replace $W^\star(\Theta)$ by the output of an iterative algorithm known to converge to a good proxy. For example, if we consider the Adaptive Moment Estimation algorithm (Adam), which is characterized in Eq. (A.1), as an optimizer, then we assume that $W^\star(\Theta)$ is close to the output after τ_{in} iterations $W_{\tau_{in}}(\Theta)$.

In contrast to denoising with analysis-sparsity regularization discussed in Chapter 2, where inner updates are available in closed form expression, inner updates here do not enjoy such property. Instead, they require AD to evaluate the gradients $\nabla_{\boldsymbol{W}_t} F_{in}$ for each Adam-based update, especially when using GNNs as the

Algorithm 2: Learning algorithm

Input: $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{A}_{obs}$
Output: $\boldsymbol{\Theta}$: $G2G$ trained weights.
Hyperparameters: η_{out}, η_{in} (learning rates), τ_{in}, τ_{out} (number of iterations), τ_{AD} (number of unrolled inner iterations, in our work $\tau_{AD} = 10$).
Randomly initialize $\boldsymbol{\Theta}$.
for $k = 1$ **to** τ_{out} **do**
  $\boldsymbol{A}_{\boldsymbol{\Theta}} \leftarrow G2G(\boldsymbol{X}, \boldsymbol{A}_{obs})$
  Randomly initialize $\boldsymbol{W}_0$.
  for $t = 1$ **to** τ_{in} **do**
    $\boldsymbol{Y}_{\boldsymbol{W}_t}(\boldsymbol{X}, \boldsymbol{A}_t) \leftarrow$ graph-based model's output.
    $\nabla_{\boldsymbol{W}_t} F_{in} \leftarrow$ *PyTorch* AD on $F_{in}(\boldsymbol{Y}, \boldsymbol{Y}_{\boldsymbol{W}_t}, \boldsymbol{A}_{\boldsymbol{\Theta}})$
    $\boldsymbol{W}_{t+1} \leftarrow$ Adam-step$(\boldsymbol{W}_t, \nabla_{\boldsymbol{W}_t} F_{in}, \eta_{in})$
    if $t == \tau_{in} - \tau_{AD}$ **then**
      Track next τ_{AD} inner updates as a function of $\boldsymbol{\Theta}$ with *Higher* package.
    end if
  end for
  $\boldsymbol{Y}_{\boldsymbol{W}_{\tau_{in}}}(\boldsymbol{X}, \boldsymbol{A}_{\boldsymbol{\Theta}}) \leftarrow$ evaluate inner model.
  $\nabla_{\boldsymbol{\Theta}} F_{out} \leftarrow$ *Higher* AD on $F_{out}(\boldsymbol{Y}, \boldsymbol{Y}_{\boldsymbol{W}_{\tau_{in}}}, \boldsymbol{A}_{\boldsymbol{\Theta}})$
  $\boldsymbol{\Theta} \leftarrow$ Adam-step$(\boldsymbol{\Theta}, \nabla_{\boldsymbol{\Theta}} F_{out}, \eta_{out})$
end for
Return: $\boldsymbol{\Theta}$, $\boldsymbol{W}_{\tau_{in}}$ (optional).

graph-based model. That said, AD is still capable of looking at the sequence of such updates as an algorithm, and evaluates the Jacobian of its output as a function of Θ, *i.e.,* $\boldsymbol{J}_{\boldsymbol{W}_{\tau_{in}}}(\boldsymbol{\Theta})$. In other words, we have a double application of AD, the first is the traditional one used to train models in most machine learning problems, while the second differentiates the resulting trained models *w.r.t.* other variables, which is not trivial. Therefore, AD can evaluate hypergradients. Additionally, we propose to use a warm start strategy where we reduce the number of unrolled iterations. Specifically, we perform τ_{in} inner iterations, but differentiate through only the last τ_{AD} iterations using AD to obtain hypergradients. This significantly reduces memory and time costs compared to unrolling all τ_{in} updates, which can be substantial when τ_{in} is large, and yet leads to better graphs than the observed ones as we empirically demonstrate.

Indeed, we use AD to approximate $\nabla_{\boldsymbol{\Theta}} F_{out}$ and apply a gradient-based algorithm, Adam, to converge to a good set of weights Θ, as described in Algorithm 2.

3.4 Memory and computation costs

Our experiments in Section 3.5 demonstrate that hypergradients obtained by tracing $\tau_{AD} = 10$ inner iterations provide a reliable estimation of the true hypergradients, as the output graphs notably outperform the observed graph. Therefore, our algorithm needs memory to place G2G weights, 10 copies of the GNN weights, or 10 copies of the optimized labels $\boldsymbol{Y}$ when using the Laplacian regularization method. That is in addition to points in the dataset, and other hyperparameters that can be ignored. Since GNN models usually don't get better results when having more than 2 layers, the memory need is of the same magnitude as the co-training method, *e.g.,* GAM (Stretcu et al., 2019), the graph attention models, and the LDS model (Franceschi et al., 2019). Regarding the computation cost, our algorithm and GAM have comparable costs when the number of co-training rounds equals the number of outer iterations. Compared to the LDS method, our framework costs less thanks to tracking just the last 10 inner iterations by AD. However, graph attention models are still the least expensive among previous algorithms.

3.5 Experiments

We design a synthetic dataset to examine the capacity of the G2G model in our framework, by trying different schemes to generate the ground-truth graph as a function of node features. On real datasets, we show that using G2G yields significant benefits over the observed graph.

Synthetic dataset: we sample *i.i.d.* latent variables $\boldsymbol{X}' \in \mathbb{R}^{n \times p}$ for points uniformly at random from $[0, 1]^p$ with $n = 256$, $p = 2$, unless otherwise specified. The Ground-Truth (GT) graph $\boldsymbol{A}_{GT}$ is then constructed s.t. an edge between points i, j has the weight

$$(\boldsymbol{A}_{GT})_{i,j} = \exp\left(-\|\boldsymbol{X}'_i - \boldsymbol{X}'_j\|_2^2 / 2\sigma^2\right).$$

We generate observed features $\boldsymbol{X}$ as a function of latent variables $\boldsymbol{X} = f(\boldsymbol{X}') \in \mathbb{R}^{n \times p}$. Each point i in the inner training set V_{tr} is labeled as follows:

$$\boldsymbol{Y}_i = \zeta(e^{-\frac{(\boldsymbol{X}'_i - \boldsymbol{a}_1)^2}{2(0.2)^2}} + e^{\frac{-(\boldsymbol{X}'_i - \boldsymbol{a}_2)^2}{2(0.2)^2}} + e^{\frac{-(\boldsymbol{X}'_i - \boldsymbol{a}_3)^2}{2(0.2)^2}}) \ ,$$

where $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$ are randomly sampled from $[0, 1]^p$, and ζ is a scaling factor such that labels lie in $[0, 1]$. By this construction, the assumption of label smoothness on the graph is met, and the Laplacian regularization Eq. (3.1) is a reasonably good strategy. To generate labels in the outer training and the validation sets V_{out}, V_{val},

respectively, we plug the labels of V_{tr} and $\boldsymbol{A}_{GT}$ in Eq. (3.1) with $\lambda = 1$, and take the solution as ground-truth labels. This way, if the G2G model learns exactly the ground-truth graph $\boldsymbol{A}_{GT}$, then the validation and outer losses are exactly equal to zero. To generate the observed graph $\boldsymbol{A}_{obs}$, we consider a classical model of random graphs:

$$(\boldsymbol{A}_{obs})_{i,j} \sim \text{Ber}\left((\boldsymbol{A}_{GT})_{i,j}\right) \ .$$

Each dataset is evenly divided into 4 subsets: inner training, outer training, validation, and unlabeled sets. σ is user-defined, our choice was s.t. the average number of edges in $\boldsymbol{A}_{obs}$ equals $n \log n$ (balance between a sparse and a complete graph). Experiments on this dataset use the Laplacian regularization as in Eq. (3.1).

Real world datasets: we demonstrate the capacity of our method on common SSL benchmark datasets for node classification: the Cora dataset which consists of 2708 nodes and 5429 edges (Lu and Getoor, 2003), the CiteSeer dataset which consists of 3327 nodes and 4732 edges (Bhattacharya and Getoor, 2007), and the PubMed dataset which consists of 19717 nodes and 44338 edges (Namata et al., 2012). These are citation datasets where points represent research publications described by means of a bag of words, and edges stand for citations. The task is to classify the unlabeled ones *w.r.t.* their main topic. From the default train/validation/test split in Yang et al. (2016); Kipf and Welling (2017), we use the training set as the inner training set V_{tr}, while we use half of the validation set as the outer training set V_{out}. The other half is kept as a validation set as in Franceschi et al. (2019).

Models: our G2G and GNN models are implemented using *PyTorch* (Paszke et al., 2019) and *PyTorch* Geometric (Fey and Lenssen, 2019), respectively. The encoder β and the regressor γ in the G2G model are an MLP of 1 hidden layer each. Likewise, the GNN has 1 hidden layer. All hidden layers in all models have 128 hidden neurons, and equipped with the *ReLu* activation function. The GNN output layer is equipped with the $softmax$ function, while the one of the G2G model with the sigmoid function.

Setup: we use Adam as the inner and outer optimizer with the default parameters of *PyTorch*, except for the learning rate set with a grid search to: $\eta_{in} = 0.1$ with Laplacian regularization, $\eta_{in} = 0.06$ with GNN models, and $\eta_{out} = 0.003$. We use *Higher* package to track unrolled inner iterations and apply the second-order automatic differentiation to compute $\nabla_{\boldsymbol{\Theta}} F_{out}$ (Grefenstette et al., 2019). We fix $\tau_{in} = 300$ with the GNN model and $\tau_{in} = 100$ with Laplacian regularization. We unroll the last $\tau_{AD} = 10$ iterations. GNN weights W and $\boldsymbol{Y}$ when using the Laplacian regularization are initialized at random after each outer iteration, using

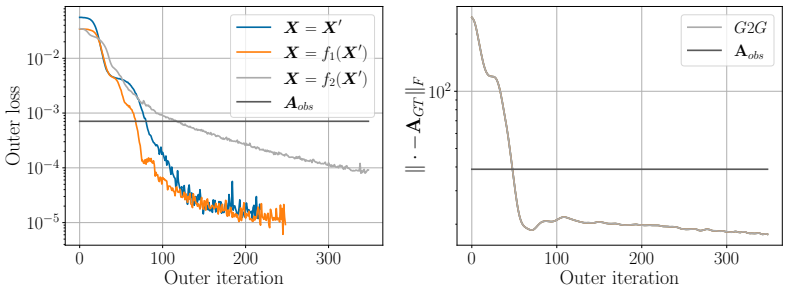


Figure 3.1 – The capacity of G2G when varying $\boldsymbol{X}$ as a function of $\boldsymbol{X}'$. We try three functions as in Section 3.5.1. Left: we plot the outer loss. Right: we plot the error between the ground-truth graph $\boldsymbol{A}_{GT}$ and the G2G's output in the case of f_2.

Xavier initialization and $\mathcal{U}(0, 1)$, respectively. The Laplacian regularization magnitude λ is fed to the algorithm in the experiments on the synthetic dataset, while set with a grid search to $\lambda = 0.01$ for Cora and CiteSeer datasets, and to $\lambda = 1$ for PubMed. We apply early stopping on the validation loss.

3.5.1 G2G capacity

To examine the expressive power of the G2G model in our proposed method, we try several options for the choice of the function f, which maps from latent variables to observed features $f : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}; \boldsymbol{X}' \mapsto \boldsymbol{X}$. Let $(x'_1, x'_2) \in \mathbb{R}^2$ be the latent variable of a point, the considered options are: $f_0(x'_1, x'_2) = (x'_1, x'_2)$, the linear transformation $f_1(x'_1, x'_2) = (x'_1 + x'_2, x'_1 - x'_2)$, and the non-linear function $f_2(x'_1, x'_2) = (\cos(\pi x'_1), \cos(\frac{\pi}{2}(x'_1 + x'_2))$. In contrast to other experiments, we tend not to feed the observed graph $\boldsymbol{A}_{obs}$ to the G2G model here. Fig. 3.1 shows convergence in the outer loss in all cases, and that the G2G model manages to notably outperform $\boldsymbol{A}_{obs}$ for Laplacian regularization in Eq. (3.1). More importantly, the G2G's output *converges to the ground-truth graph* even with the non-linear mapping f_2. As G2G doesn't have access to $\boldsymbol{A}_{obs}$ during training, this implies the capacity of this framework processing only $\boldsymbol{X}$ to extract $\boldsymbol{X}'$, and then capture the rules governing the construction of the graph $\boldsymbol{A}_{GT}$, while being only guided by maximizing regression performance in Eq. (3.3). This proves the power of the G2G model, and the efficiency in computing hypergradients using the package *Higher*, as well as indicates some information-preservation phenomenon when constructing the latent position graph and its labels, which will be analyzed theoretically in future work.

Table 3.1 – Benchmark against a GNN model operating on the given graph and the GAM model on real-world datasets. We report the classification test accuracy. The subscript of models indicates the number of hidden neurons in each hidden layer. Results of GAM method are reported from the according paper.

Model	Dataset		
	Cora	CiteSeer	PubMed
GNN_{128}	77.0	67.0	75.2
$GNN_{128} + GAM$	84.8	72.2	81.0
$GNN_{128} + G2G$	79.6	71.8	79.2
$Laplacian + G2G$	78.9	54.7	70.3

3.5.2 Results on real-world datasets

We conduct our experiments on Cora, CiteSeer and PubMed datasets. During training, we augment the given graph by sampling 1000 edges from all possible edges at every iteration in the outer problem. We keep up to 10000 edges that were assigned the highest weights by the G2G model. This way, we don't just denoise the given edges, but we are likely to find more helpful edges, while avoiding processing the complete graph with these datasets, which is memory consuming and time costly. We run our algorithm two times on these datasets, where we use the Laplacian regularization in the inner problem in one, and a GNN model in the second time. As seen in Table 3.1, GNNs outperform the Laplacian regularizer when used in the inner problem, which is expected due to the message passing model in GNNs. Besides, we see that our algorithm with a GNN in the inner problem yields significant improvement over the GNN model operating on the given graph. On the other hand, the Graph Agreement Model (GAM) produces higher accuracy on the three datasets, thus our framework is not state-of-the-art.

Chapter 4

Hypergradient scarcity in graph learning

We stated in the previous chapter that edge refinement by optimizing observed edge weights via the bilevel framework does not produce graphs of quality. Instead, we proposed to incorporate metric learning in the bilevel framework using G2G models. In this chapter, which includes the theoretical contribution of this thesis, we theoretically support our statement. Precisely, we investigate a phenomenon in graph learning under the SSL setting that we refer to as *gradient scarcity*, which deteriorates the quality of the learned graph. In Section 4.1, we introduce the context in which this phenomenon was first identified, which involves optimizing for observed edge weights and the weights of a GNN using joint optimization to minimize a labeling loss. Under this setting, edges between unlabeled nodes that are far from labeled ones receive zero gradients. Subsequently, we formulate the problem of edge refinement with bilevel optimization in Section 4.2. Next, we pose the research questions we tackle in this chapter, mainly examining this issue within the bilevel optimization setting and with other graph-based models. We then outline our contributions. In Section 4.4 we review related works that identified this issue or used bilevel optimization for graph learning. In Section 4.5, we give a precise mathematical characterization of this phenomenon, and prove that it also emerges in *bilevel* optimization, where additional dependency exists between the parameters of the problem. While gradient scarcity with GNNs occurs due to their finite receptive field, we show in Section 4.6 that it also occurs with the Laplacian regularization model, in the sense that hypergradients amplitude decreases exponentially with distance to labeled nodes. In Section 4.7, we study several solutions to alleviate this issue including metric learning using G2G models, graph regularization, or optimizing on a larger graph than the original one with

a reduced diameter. Finally in Section 4.8, we present our empirical results on synthetic and real datasets that validate our analysis and prove the efficiency of the proposed solutions.

The content of this chapter has been submitted for publication in our article Ghanem et al. (2023b), and is currently under revision.

4.1 Understanding gradient scarcity: context and observations

As discussed in Section 1.4.2, a mainstream approach in graph learning for graph-based SSL is to optimize the graph by means of optimizing the performance in the downstream task. This involves generating a graph that, when used by the graph-based model, minimizes some loss on labeled nodes. However, the graph-based model itself requires an optimization process on its parameters to minimize the classification or the regression loss for instance. Therefore, both the graph learner and the graph-based model need to learn by minimizing the "same" loss. There are two methodologies to formulate this mathematically, namely *joint* and *bilevel* optimization, which both are usually solved by gradient-based algorithms.

In joint optimization, one objective function F is defined as a function of both the graph $\boldsymbol{A}$ and the graph-based model. Let us denote by W the learnable parameters of the graph-based model $\boldsymbol{Y}_W$, then joint optimization minimizes:

$$\min_{\boldsymbol{A},W} F = \frac{1}{|V_{tr}|} \sum_{i \in V_{tr}} \ell\Big(\big(\boldsymbol{Y}_W(\boldsymbol{X}, \boldsymbol{A})\big)_i, (\boldsymbol{Y}_{obs})_i \Big) \ ,$$

where ℓ is a smooth loss function and V_{tr} is the training set of labeled nodes. In this scenario, both $\boldsymbol{A}$ and W are simultaneously updated in each iteration of the gradient-based algorithm. Another scheme of updates, called *alternating* optimization, that can be employed in this scenario fixes one object while the other is updated in one iteration, and vice versa in the next iteration. Nonetheless, alternating optimization is not commonly used in the graph learning paradigm.

For Graph Neural Networks (GNNs) as the graph-based model, Fatemi et al. (2021) show that directly learning the weights of observed edges with joint optimization leads to *gradient scarcity*. This phenomenon refers to the fact that connections between unlabeled nodes "far" from the labeled ones receive *zero gradients*, *i.e.,* they receive no supervision during the optimization and are not learned. This is due to the finite receptive field (depth) of message-passing GNNs. Putting all together, gradient scarcity was identified under the edge refinement setting when learning edge weights and a GNN model using joint optimization.

In this work, we focus on bilevel optimization and prove that gradient scarcity also occurs for GNNs, despite additional dependency between the parameters in the bilevel setting. We also prove that this issue emerges with other graph-based models, including the Laplacian regularization, which, unlike GNNs, has an infinite receptive field.

Next, we formulate the problem of learning the weights of observed edges with bilevel optimization. Then, we state the research questions we tackle and the contributions of this chapter.

4.2 Edge refinement with bilevel optimization

In bilevel optimization, we consider the case where the graph objective function is a function of the trained graph-based model. Using a second set of labeled nodes $V_{out} \subset V$ distinct from V_{tr}, the bilevel optimization is cast as

$$\boldsymbol{A}^\star \in \underset{\boldsymbol{A}\in\mathcal{A}}{\arg\min}\, F_{out}(\boldsymbol{A}) = \tfrac{1}{|V_{out}|} \sum_{i\in V_{out}} \ell(\boldsymbol{Y}(\boldsymbol{A})_i, (\boldsymbol{Y}_{obs})_i), \tag{4.1}$$

such that $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{GNN}(\boldsymbol{X}, \boldsymbol{A})$ in the scenario of a GNN as the graph-based model, or $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A})$ in the Laplacian regularization scenario. The model we consider in this chapter for the set of admissible adjacency matrices $\mathcal{A}$ is *edge refinement*: the learned adjacency matrix has the same zero-pattern as the observed adjacency matrix:

$$\mathcal{A} = \{\boldsymbol{A} \in [a, b]^{n\times n} | \boldsymbol{A}_{ij} = 0 \text{ when } (\boldsymbol{A}_{obs})_{ij} = 0\},$$

for some bounds a, b. Recall that the complexity in this setting is proportional to the number of edges, generally less than quadratic in the number of nodes n as graphs are usually sparse, which makes edge refinement more scalable than the full learning setting.

4.3 Research questions and contribution

Previous works observed gradient scarcity when learning the graph and a GNN model with joint optimization. This issue is due to the finite receptive field of GNNs, since a k-layer GNN computes the label of a node using information only from r-hop far nodes with $r \leq k$. This label is then not a function of edges connecting nodes outside of this neighborhood, and the term in the learning loss corresponding to this label returns null gradients on those distant edges. However, it is not straightforward how to extend this argument to the bilevel setting. Specifically, the previous discussion is in the context of joint optimization which assumes

that the trained weights of the GNN after many gradient-based updates do not depend on the adjacency matrix $\boldsymbol{A}$, which is not the case in bilevel optimization. Moreover, if the problem holds in this setting, the roles of V_{tr} and V_{out} need to be clarified. Another question is if this problem is mitigated by resorting to graph-based models with infinite receptive field, *e.g.,* the Laplacian regularization.

In this chapter, **we prove that hypergradient scarcity occurs under the bilevel optimization setting when adopting GNNs as a graph-based model.** We show that using a k-layer GNN induces null hypergradients on edges between nodes at least k-hop from labeled nodes in $V_{tr} \cup V_{out}$. **For the Laplacian regularization, we prove that the problem persists**, as hypergradients are exponentially damped with distance from labeled nodes. **We empirically validate our findings**. Then, we test three possible strategies to solve this issue: metric learning with G2G models, graph regularization and refining a power of the observed adjacency matrix. Furthermore, we empirically **distinguish between hypergradient scarcity and overfitting**, in the sense that solving the former does not necessarily resolve the latter. To the best of our knowledge, this is the first work that mathematically tackles the gradient scarcity problem for bilevel optimization of graphs, and examines the phenomenon for models with infinite receptive field.

In the next section, we review the relevant literature with a special focus on works that tackled graph learning with bilevel optimizations and the ones that identified gradient scarcity. Then we prove the theorems that support our claims.

4.4 Related work

As reviewed in Section 1.5, bilevel optimization applications are numerous, including multi-task and meta learning (Bennett et al., 2006; Flamary et al., 2014; Franceschi et al., 2018). Graph learning, on the other hand, plays a crucial role in handling real-world graphs that often exhibit noisy edges. To learn better graphs, various methods have been proposed in the literature; however, only a few of them leverage bilevel optimization.

One work in this direction is Franceschi et al. (2019). The authors learn the parameters of Bernoulli probability distributions over independent random edges. The problem is similarly framed as a bilevel optimization, where these parameters are optimized to minimize the GNN's validation loss. Similar to Eq. (4.1) with full learning of $\boldsymbol{A}$, this method includes learning n^2 parameters which limits scalability. Instead of learning observed edge weights, Wan and Kokel (2021) sparsify the observed graph while keeping it connected by means of minimizing the validation

loss of the GNN model. However, resulted graphs do not necessarily outperform the observed one.

Another line of research adopts the joint optimization scenario. One instance is the work of Wang and Leskovec (2020), where authors propose to alleviate overfitting resulted from learning the GNN parameters and edge weights together by using the Label Propagation model (LP) (Zhu, 2005) to regularize the graph. The proposed framework produces state-of-the-art results on node classification tasks. Another example is attention mechanisms, where edge weights are re-evaluated after each GNN layer considering the similarity between node representations, *i.e.,* edge refinement. The similarity criterion can be user-defined like the dot product (Luong et al., 2015; Vaswani et al., 2017), learned locally at each layer by a single-layer feed-forward network (Veličković et al., 2018), or a combination of both schemes (Kim and Oh, 2021).

Other graph learning methods do not involve the joint optimization nor the bilevel optimization schemes. In Stretcu et al. (2019) for example, a state-of-the-art method referred to as Graph Agreement Model (GAM) is proposed to learn graphs by penalizing the absence of an edge between nodes with the same label.

The gradient scarcity problem was studied in Fatemi et al. (2021) where the authors looked at this problem with the intuition that learning a graph in SSL problems is done to improve performance in the downstream task, thus optimizing both requires such supervision that is not available in small labeled subsets. Then, for downstream tasks adopting a k-layer GNN model (with $k = 2$ in their case), they identified what they refer to as the *supervision starvation problem*, which states that edges between unlabeled nodes do not receive any supervision if they are at least 2-hop from labeled nodes. They quantify the starvation for the special case of Erdös-Rényi graphs. Note that gradient scarcity and supervision starvation refer to the same phenomenon.

This issue cannot be resolved by adding more layers to the GNN as this will increase its complexity on one hand, which means more data and labels are needed, and due to the oversmoothing issue on the other hand (Keriven, 2022). To mitigate this issue and provide more supervision on the graph level, authors make use of the assumption that a good graph does not only perform well in labeling nodes, but also in denoising node features. Therefore, they regularize the learned graph by a contrastive loss (Liu et al., 2021b; Wu et al., 2021; Liu et al., 2022a), which evaluates its denoising performance. Overall, this results in a joint optimization problem.

That said, authors implicitly assumed no dependence between the GNN weights and the graph when identifying gradient scarcity, which is the case in joint/alternating

optimization schemes. To the best of our knowledge, this issue has not yet been studied for the bilevel optimization setting. Moreover, it is not clear if this problem is resolved with graph-based models with infinite receptive field, *e.g.,* the Laplacian regularization. We treat both these topics in this chapter.

In Liu et al. (2022b), authors state that optimizing both the graph and a GNN model under the supervision of a classification task introduces reliance on available labels, bias in the edge distribution and even reduce the span of potential application tasks. Still, this statement is not accompanied with a theoretical justification, especially regarding the first two consequences. To overcome this problem, authors suggested to avoid label-based graph optimization, and proposed an *unsupervised* graph learning framework based on contrastive learning (Liu et al., 2021b). Although the unsupervised framework proved effective and competed state-of-the-art methods, we believe that labels contain informative knowledge that is not exploited when deploying unsupervised learners, and that better results are obtained by getting the best of both worlds.

4.5 The GNNs scenario

In this section, we consider the bilevel optimization (4.1) in the *edge refinement* setting, *i.e.,* we optimize the weight of every existing edge in $\boldsymbol{A}_{obs}$, and the GNN case $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{GNN}(\boldsymbol{X}, \boldsymbol{A}) = \boldsymbol{Y}_{W^\star}(\boldsymbol{X}, \boldsymbol{A})$. For the convenience of the reader, we recall the GNN model we adopt in this work. This model propagates the first layer $\boldsymbol{X}^{[0]} = \boldsymbol{X}$ as follows

$$\boldsymbol{X}^{[l]} = \phi(\boldsymbol{X}^{[l-1]}\boldsymbol{W}_1^{[l]} + \boldsymbol{A}\boldsymbol{X}^{[l-1]}\boldsymbol{W}_2^{[l]} + \mathbf{1}_n(\boldsymbol{b}^{[l]})^\top) \ , \tag{4.2}$$

where $W = \{\boldsymbol{W}_1^{[l]}, \boldsymbol{W}_2^{[l]}, \boldsymbol{b}^{[l]}\}_{l=1}^k$ is the set including the model weights, ϕ is a non-linear function applied element-wise, and the output $\boldsymbol{Y}_W(\boldsymbol{X}, \boldsymbol{A}) = \boldsymbol{X}^{[k]}$ is obtained after k rounds of message passing. We denote by $\boldsymbol{W}$ the vectorized version of W.

First, we examine the joint/alternating optimization schemes, where the dependency between W and $\boldsymbol{A}$ is dropped, *i.e.,* $\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A}) = \mathbf{0}$, and prove the existence of the problem for a generic number of layers k, similar to Fatemi et al. (2021). For the bilevel optimization setting, we then prove that the optimized weights $\boldsymbol{W}^\star$ are not a function of edges connecting nodes at least k-hop from nodes in V_{tr}. After that, we conclude that hypergradient scarcity holds in the bilevel setting for edges connecting nodes at least k-hop from nodes in the *union* $V_{tr} \cup V_{out}$.

4.5.1 Scarcity with joint or alternating optimization

In this first result, we will assume that the weights $\boldsymbol{W}$ do not depend on $\boldsymbol{A}$, as is the case in joint/alternating minimization, and show gradient scarcity by analyzing $\boldsymbol{Y}_{\boldsymbol{W}}(\boldsymbol{X}, \boldsymbol{A})$.

Theorem 4.5.1. *Let $\boldsymbol{Y}_{\boldsymbol{W}} = \boldsymbol{Y}_{\boldsymbol{W}}(\boldsymbol{X}, \boldsymbol{A})$ be the output of a k-layer GNN parameterized by $\boldsymbol{W}$. Let i, j, u be such that nodes i, j are at least k-hop from node u. Assume that $\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{A}_{i,j}} = \boldsymbol{0}$. Then:*

$$\frac{\partial (\boldsymbol{Y}_{\boldsymbol{W}})_u}{\partial \boldsymbol{A}_{i,j}} = 0 \ . \tag{4.3}$$

Proof. The proof is done by induction on k. For $k = 1$, this is indeed the case since $\boldsymbol{X}^{[0]} = \boldsymbol{X}$ does not depend on $\boldsymbol{A}$, and that $\boldsymbol{A}_{i,j}$ does not belong to the row $\boldsymbol{A}_{u,:}$ which is the only row in $\boldsymbol{A}$ that contributes in the value $(\boldsymbol{X}^{[1]})_{u,:}$.

Assume that the statement is true for some arbitrary positive integer k, we show that it is also true for a $(k+1)$-layer GNN. If i, j are at least $(k+1)$-hop from u, then clearly they are at least k-hop far from it too. Thus from the induction assumption, we have that $(\boldsymbol{X}^{[k]})_{u,:}$ is independent of $\boldsymbol{A}_{i,j}$. Also, $\boldsymbol{W}_1^{[k+1]}$ does not depend on $\boldsymbol{A}_{i,j}$ since we assume $\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{A}_{i,j}} = \boldsymbol{0}$. Therefore, $(\boldsymbol{X}^{[k]}\boldsymbol{W}_1^{[k+1]})_{u,:}$ in (4.2) does not depend on $\boldsymbol{A}_{i,j}$ too.

In a similar way, if i, j are at least $(k+1)$-hop from u, then they are at least k-hop far from any of its neighbors v where $\boldsymbol{A}_{u,v} \neq 0$. Therefore, for all v, $\boldsymbol{A}_{u,v} \neq 0$, then $\frac{\partial (\boldsymbol{X}^{[k]})_{v,:}}{\partial \boldsymbol{A}_{i,j}} = \boldsymbol{0}$. Moreover, $\frac{\partial \boldsymbol{W}_2^{[k+1]}}{\partial \boldsymbol{A}_{i,j}} = \boldsymbol{0}$ since we assume $\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{A}_{i,j}} = \boldsymbol{0}$. This makes $(\boldsymbol{A}\boldsymbol{X}^{[k]}\boldsymbol{W}_2^{[k+1]})_{u,:} = \boldsymbol{A}_{u,:}\boldsymbol{X}^{[k]}\boldsymbol{W}_2^{[k+1]}$ in (4.2) independent of $\boldsymbol{A}_{i,j}$. This concludes the proof, as $\frac{\partial (\boldsymbol{Y}_{\boldsymbol{W}})_u}{\partial \boldsymbol{A}_{i,j}} = \frac{\partial (\boldsymbol{X}^{[k+1]})_u}{\partial \boldsymbol{A}_{i,j}} = \boldsymbol{0}$. □

4.5.2 Gradient of the optimized weights

Theorem 4.5.1 assumes that $\boldsymbol{W}$ is not a function of the edge $\boldsymbol{A}_{i,j}$, and states, in such case, that edges between nodes at least k-hop from the training nodes used to optimize the graph (V_{out} in our case) receive no supervision. However, $\boldsymbol{W}$ may depend on $\boldsymbol{A}$ after the first outer iteration in the bilevel optimization scenario. The next theorem shows that gradient scarcity still occurs in the bilevel optimization framework, as the "optimal" weights used in practice are the result of a gradient-based algorithm. More precisely, we consider a sequence

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - Q_t(\boldsymbol{W}_t, \nabla_{\boldsymbol{W}_t} F_{in}) \ , \tag{4.4}$$

where Q_t is a smooth function. Note that $\boldsymbol{W}_t$ does not necessarily converge towards the true optimal point $\boldsymbol{W}^\star$.

Theorem 4.5.2. *Let $\boldsymbol{A}$ be an input graph to a k-layer GNN with weights $\boldsymbol{W}$, and $\boldsymbol{W}_t$ be the output obtained by optimizing (1.5) for $\boldsymbol{W}$ using a gradient-based iterates sequence. Let i, j be nodes that are at least k-hop from any node in V_{tr}. Then, for all $t \in \mathbb{N}$,*

$$\frac{\partial \boldsymbol{W}_t(\boldsymbol{A})}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0} \ . \tag{4.5}$$

Proof. The proof is carried out by induction on the iteration index t of the gradient-based optimizer. Denote by F_{in} the objective function in (1.5). For $t = 0$, $\boldsymbol{W}_0$ is the initialization of $\boldsymbol{W}$ which is usually random and does not depend on $\boldsymbol{A}$. For $t \geq 0$, we assume that $\frac{\partial \boldsymbol{W}_t}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$ and prove this must be true for $t+1$. By the chain rule, proving that $\frac{\partial(\nabla_{\boldsymbol{W}_t} F_{in})}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$ is sufficient to complete the proof. The gradient $\nabla_{\boldsymbol{W}_t} F_{in}$ writes:

$$\nabla_{\boldsymbol{W}_t} F_{in} = \frac{1}{|V_{tr}|} \sum_{u \in V_{tr}} \nabla_{\boldsymbol{W}_t} \ell\Big(\big(\boldsymbol{Y}_{\boldsymbol{W}_t}(\boldsymbol{X}, \boldsymbol{A})\big)_u, (\boldsymbol{Y}_{obs})_u \Big) \ .$$

For all $u \in V_{tr}$, the term $\nabla_{\boldsymbol{W}_t} \ell\Big(\big(\boldsymbol{Y}_{\boldsymbol{W}_t}(\boldsymbol{X}, \boldsymbol{A})\big)_u, (\boldsymbol{Y}_{obs})_u \Big)$ is a function of $\boldsymbol{W}_t$ and $\big(\boldsymbol{Y}_{\boldsymbol{W}_t}(\boldsymbol{X}, \boldsymbol{A})\big)_u$. But $\frac{\partial \boldsymbol{W}_t}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$ from the induction assumption, and, given that, we have $\frac{\partial \big(\boldsymbol{Y}_{\boldsymbol{W}_t}(\boldsymbol{X}, \boldsymbol{A})\big)_u}{\partial \boldsymbol{A}_{i,j}} = 0$ from Theorem 4.5.1. Thus, we have for all $u \in V_{tr}$, $\frac{\partial}{\partial \boldsymbol{A}_{i,j}} \nabla_{\boldsymbol{W}_t} \ell\Big(\big(\boldsymbol{Y}_{\boldsymbol{W}_t}(\boldsymbol{X}, \boldsymbol{A})\big)_u, (\boldsymbol{Y}_{obs})_u \Big) = \mathbf{0}$. This concludes the proof of (4.5) as it gives $\frac{\partial(\nabla_{\boldsymbol{W}_t} F_{in})}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$. □

4.5.3 Hypergradient scarcity

Finally, we put the two previous results together. The next theorem states that within the bilevel optimization framework, edges between nodes at least k-hop from nodes in $V_{tr} \cup V_{out}$ receive no supervision.

Theorem 4.5.3. *Let $\boldsymbol{Y}_{\boldsymbol{W}}$ be a k-layer GNN. Assume that the inner optimization problem is solved with a gradient-based algorithm (4.4). Then, for any pair of nodes i, j at least k-hop from nodes in $V_{out} \cup V_{tr}$, we have $\frac{\partial F_{out}}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$.*

Proof. Directly from Theorem 4.5.2 we have that $\frac{\partial \boldsymbol{W}_t(\boldsymbol{A})}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$ since i, j are at least k-hop from nodes in V_{tr}. This makes it possible to apply Theorem 4.5.1 to get that $\forall u \in V_{out}$; $\frac{\partial (\boldsymbol{Y}_{\boldsymbol{W}_t})_u}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$, as i, j are at least k-hop from nodes in V_{out} and

$\frac{\partial \boldsymbol{W}_t(\boldsymbol{A})}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$. This concludes the proof as F_{out} penalizes the labeling error only on nodes in V_{out}. □

Theorem 4.5.3 shows that the hypergradient scarcity problem emerges when solving edge refinement tasks: if two nodes are at least k-hop from nodes in $V_{out} \cup V_{tr}$ in $\boldsymbol{A}_{obs}$, the edge in between receives no hypergradients. In Section 4.7, we will examine several strategies to mitigate this phenomenon.

4.6 The Laplacian regularization scenario

In the previous section, we have seen how the finite receptive field of GNNs directly induces the gradient scarcity problem. We now examine hypergradient scarcity when $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A})$ as in (1.1) with the Laplacian regularization (1.2). Indeed, in this case the inner problem (1.1) does not have a finite receptive field, in the sense that in general $\frac{\partial \boldsymbol{Y}(\boldsymbol{A})}{\partial \boldsymbol{A}_{ij}} \neq 0$ for all i, j, unlike the GNN case as proven by Theorem 4.5.1.

Surprisingly, we show that hypergradient scarcity still occurs in some sense. More precisely, we prove that the magnitude of hypergradients decreases exponentially with the sum of the distance to V_{tr} and the distance to V_{out}.

We consider the case where the downstream task is a regression problem, *i.e.,* ℓ in Eqs. (1.1) and (4.1) is the MSE loss function. Let $\boldsymbol{S}_{in} \in \mathbb{R}^{n \times n}$ be the diagonal selection matrix whose entries equal 1 if the corresponding node is in V_{tr} and 0 otherwise, the solution $\boldsymbol{Y}(\boldsymbol{A})$ enjoys a closed-from expression:

$$\boldsymbol{Y}(\boldsymbol{A}) = \left(\tilde{\boldsymbol{S}}_{in} + \lambda \tilde{\boldsymbol{L}}\right)^{-1} \tilde{\boldsymbol{S}}_{in} \boldsymbol{Y}_{obs} \ ,$$

where $\tilde{\boldsymbol{S}}_{in} = \frac{\boldsymbol{S}_{in}}{|V_{tr}|}$ and $\tilde{\boldsymbol{L}} = \frac{\boldsymbol{L}}{|E|}$. For simplicity from now on, we denote $\boldsymbol{B} = \tilde{\boldsymbol{S}}_{in} + \lambda \tilde{\boldsymbol{L}}$. Then, we write $\boldsymbol{Y}(\boldsymbol{A})$ as:

$$\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{B}^{-1} \tilde{\boldsymbol{S}}_{in} \boldsymbol{Y}_{obs} \ . \tag{4.6}$$

It is well-defined thanks to the following result.

Lemma 4.6.1. *Assume that the graph is connected. The eigenvalues μ_i of $\boldsymbol{B}$ satisfy, for all i:*

$$0 < \mu_{\min} \leq \mu_i \leq \mu_{\max} \leq \frac{1}{|V_{tr}|} + 2\lambda \ . \tag{4.7}$$

Given that, we now state the main result of this section.

Theorem 4.6.2. *Let nodes i, j be at least k-hop from V_{out}, and q-hop from V_{tr}. Then we have:*

$$\left|\frac{\partial F_{out}}{\partial \boldsymbol{A}_{ij}}\right| \lesssim \lambda \frac{\sqrt{|V_{out}|} + \mu_{\min}\sqrt{|V_{tr}|}|V_{out}|}{\mu_{\min}^3 |V_{tr}||E|} y_\infty^2 (1-\mu)^{q+k} \quad , \tag{4.8}$$

where $\mu = \frac{\mu_{\min}}{\mu_{\max}}$ and $y_\infty = \|\boldsymbol{Y}_{obs}\|_\infty$.

Since both $\mu_{\min}, \mu_{\max}$ are strictly positive, as shown in the proof in Section 4.6.3, then $0 < 1 - \mu < 1$. Therefore, Theorem 4.6.2 states that the magnitude of the hypergradient is *exponentially* damped in a speed that is at least proportional to $(1-u)^{q+k}$, leading to a form of hypergradient scarcity.

The rest of this section is dedicated to proving Lemma 4.6.1 and Theorem 4.6.2. We first express $\boldsymbol{Y}(\boldsymbol{A})$ as a Neumann series, then we bound the derivative of terms in the resulted series, and by extension the gradient of F_{out}.

4.6.1 Proof of Lemma 4.6.1 and Neumann series expansion

In the first step, we re-write the inverse of $\boldsymbol{B}$ using Neumann series. We first need to prove that $\|\boldsymbol{I} - \boldsymbol{B}\| < 1$ (see *e.g.,* Stewart (1998)), where $\boldsymbol{I} \in \mathbb{R}^{n \times n}$ is the identity matrix. Remark that the eigenvalues of $\boldsymbol{I} - \boldsymbol{B}$ are $1 - \mu_i$ where $\mu_1, \ldots, \mu_n$ are the eigenvalues of $\boldsymbol{B}$. Assuming the graph is connected, the ordered eigenvalues $\{\nu_i\}_{i=0}^n$ of $\tilde{\boldsymbol{L}}$ satisfy:

$$0 = \nu_1 < \nu_2 \leq \ldots \leq \nu_n \leq 2 \ . \tag{4.9}$$

The last inequality holds because $\|\boldsymbol{L}\| \leq 2d_{\max} \leq 2|E|$, where $d_{\max}$ is the maximum degree of the graph. Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ be the eigenvectors of $\tilde{\boldsymbol{L}}$, where $\boldsymbol{u}_1 \propto \boldsymbol{1}_n$ is associated to 0.

Proof of Lemma 4.6.1. We have $\|\tilde{\boldsymbol{S}}_{in}\| \leq 1/|V_{tr}|$ and $\|\tilde{\boldsymbol{L}}\| \leq 2$ so by a triangular inequality the upper bound is proved.

Using the eigendecomposition of $\tilde{\boldsymbol{L}}$ and recalling that $\nu_1 = 0$, for any $\boldsymbol{x} \in \mathbb{R}^n$:

$$\begin{aligned} \boldsymbol{x}^\top \boldsymbol{B} \boldsymbol{x} &= \lambda \boldsymbol{x}^\top \tilde{\boldsymbol{L}} \boldsymbol{x} + \boldsymbol{x}^\top \tilde{\boldsymbol{S}}_{in} \boldsymbol{x} \\ &= \lambda \sum_{i=2}^{n} (\boldsymbol{x}^\top \boldsymbol{u}_i)^2 \nu_i + \frac{\sum_{i \in V_{tr}} \boldsymbol{x}_i^2}{|V_{tr}|} \end{aligned}$$

which, minimized over the unit sphere, gives the expression of $\mu_{\min}$. It is immediate that $\mu_{\min} \geq 0$. We prove that this value is strictly positive. Indeed, $\boldsymbol{x}^\top \boldsymbol{B} \boldsymbol{x} = 0$ implies that $\boldsymbol{x}^\top \boldsymbol{S}_{in} \boldsymbol{x} = 0$ and therefore $\boldsymbol{x}_i = 0$ for $i \in V_{tr}$, but also that $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{0}$ and therefore that $\boldsymbol{x} \propto \boldsymbol{1}_n$, which implies that $\boldsymbol{x} = 0$. □

Let $\tilde{\boldsymbol{B}} = \boldsymbol{B}/\mu_{\max}$, with eigenvalues

$$0 \leq 1 - \mu_i/\mu_{\max} \leq 1 - \mu < 1 \ ,$$

where $\mu = \frac{\mu_{\min}}{\mu_{\max}}$. Using Neumann expansion, $\tilde{\boldsymbol{B}}^{-1}$ writes:

$$\tilde{\boldsymbol{B}}^{-1} = \sum_{r=0}^{\infty} (\boldsymbol{I} - \tilde{\boldsymbol{B}})^r \Rightarrow \boldsymbol{Y}(\boldsymbol{A}) = \sum_{r=0}^{\infty} (\boldsymbol{I} - \tilde{\boldsymbol{B}})^r \mu_{\max}^{-1} \tilde{\boldsymbol{S}}_{in} \boldsymbol{Y}_{obs} \ . \tag{4.10}$$

We denote by $\boldsymbol{T}_r$ the r-th term in $\boldsymbol{Y}(\boldsymbol{A})$:

$$\boldsymbol{T}_r = (\boldsymbol{I} - \tilde{\boldsymbol{B}})^r \mu_{\max}^{-1} \tilde{\boldsymbol{S}}_{in} \boldsymbol{Y}_{obs} \ . \tag{4.11}$$

Note that since $\|\boldsymbol{S}_{in} \boldsymbol{Y}_{obs}\| \leq \sqrt{|V_{tr}|} \|\boldsymbol{Y}_{obs}\|_\infty$, we have:

$$\|\boldsymbol{T}_r\| \leq \frac{\nu^r y_\infty}{\mu_{\max} \sqrt{|V_{tr}|}} \ , \tag{4.12}$$

where $y_\infty = \|Y_{obs}\|_\infty$ and $\nu = 1 - \mu$. Similarly, $\|\boldsymbol{Y}(\boldsymbol{A})\| \leq \frac{y_\infty}{\mu_{\min} \sqrt{|V_{tr}|}}$. Moreover, since $\boldsymbol{I} - \tilde{\boldsymbol{B}}$ has the same zero-pattern than $\boldsymbol{A}$ (except on the diagonal), if u is more than r hops from V_{tr}, we get $(\boldsymbol{T}_r)_u = 0$.

4.6.2 Gradient of $(\boldsymbol{T}_r)_u$

In the second step, we derive the formula of the gradient of $(\boldsymbol{T}_r)_u$ *w.r.t.* $\boldsymbol{A}$, and derive a bound on its magnitude as a function of r, q the distance to V_{tr}, and k the distance to V_{out}. For $r > 0$, the gradient of the u-th coefficient in $\boldsymbol{T}_r$ *w.r.t.* $\boldsymbol{I} - \tilde{\boldsymbol{B}}$ is:

$$\begin{aligned} \nabla_{\boldsymbol{I} - \tilde{\boldsymbol{B}}} (\boldsymbol{T}_r)_u = \sum_{h=1}^{r} \Big(\big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{r-h} \big)_{u,:} \Big)^\top \\ \times \big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{h-1} \mu_{\max}^{-1} \tilde{\boldsymbol{S}}_{in} \boldsymbol{Y}_{obs} \big)^\top, \end{aligned}$$

by the product rule of differentiation, and we have

$$\nabla_{\boldsymbol{I} - \tilde{\boldsymbol{B}}} (\boldsymbol{T}_r)_u = \sum_{h=1}^{r} \Big(\big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{r-h} \big)_{u,:} \Big)^\top (\boldsymbol{T}_{h-1})^\top \ .$$

Using that $\boldsymbol{I} - \tilde{\boldsymbol{B}} = \boldsymbol{I} - \frac{1}{\mu_{\max}} (\tilde{\boldsymbol{S}}_{in} + \lambda \tilde{\boldsymbol{L}})$, we have

$$\begin{aligned} \nabla_{\tilde{\boldsymbol{L}}} (\boldsymbol{T}_r)_u &= -\frac{\lambda}{\mu_{\max}} \nabla_{\boldsymbol{I} - \tilde{\boldsymbol{B}}} (\boldsymbol{T}_r)_u \\ &= -\frac{\lambda}{\mu_{\max}} \sum_{h=1}^{r} \Big(\big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{r-h} \big)_{u,:} \Big)^\top (\boldsymbol{T}_{h-1})^\top \ . \end{aligned} \tag{4.13}$$

And finally, by deriving $\tilde{\boldsymbol{L}}$ *w.r.t.* $\boldsymbol{A}_{ij}$:

$$\begin{aligned}\frac{\partial(\boldsymbol{T}_r)_u}{\partial \boldsymbol{A}_{ij}} = -\frac{\lambda}{|E|\mu_{\max}} \sum_{h=1}^{r} &\big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{r-h}\big)_{ui}(\boldsymbol{T}_{h-1})_i \\ &+ \big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{r-h}\big)_{uj}(\boldsymbol{T}_{h-1})_j \\ &- \big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{r-h}\big)_{uj}(\boldsymbol{T}_{h-1})_i \\ &- \big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{r-h}\big)_{ui}(\boldsymbol{T}_{h-1})_j \ ,\end{aligned} \tag{4.14}$$

which allows us to prove the following.

Lemma 4.6.3. *Let i, j, u such that: i, j are at least k-hop from u, and at least q-hop from V_{tr}. Then:*

$$\left|\frac{\partial(\boldsymbol{T}_r)_u}{\partial \boldsymbol{A}_{ij}}\right| \leq \begin{cases} 0 & \text{if } q + k > r \\ \frac{4\lambda y_\infty}{|E|\mu_{\max}^2\sqrt{|V_{tr}|}}(r - q - k)\nu^{r-1} & \text{otherwise.} \end{cases} \tag{4.15}$$

Proof. Recall that $(\boldsymbol{T}_r)_u = 0$ if u is more than r-hop from V_{tr}. Similarly, $((\boldsymbol{I} - \tilde{\boldsymbol{B}})^r)_{ui} = 0$ if u and i are more than r-hop from each other. Hence, the term $\big((\boldsymbol{I} - \tilde{\boldsymbol{B}})^{r-h}\big)_{ui}(\boldsymbol{T}_{h-1})_i$ appearing in (4.14) is 0 if $r - h < k$ or $h - 1 < q$, and bounded by $(\mu_{\max}\sqrt{|V_{tr}|})^{-1}\nu^{r-1}y_\infty$ otherwise. Similarly for the other terms, so the sum in (4.14) runs over the indices h that satisfy $q + 1 \leq h \leq r - k$, which is either none if $q + 1 + k > r$, or $r - q - k$ terms otherwise, which concludes the proof. □

4.6.3 Proof of Theorem 4.6.2

We finally examine the hypergradient, and prove an exponential damping rate of its magnitude with the cumulative distance to V_{tr} and V_{out} (the sum of both distances). Considering $F_{out} = \|\boldsymbol{S}_{out}(\boldsymbol{Y}(\boldsymbol{A}) - \boldsymbol{Y}_{obs})\|^2$, where $\boldsymbol{S}_{out}$ is the diagonal selection matrix whose diagonal entries equal 1 if the corresponding node is in V_{out} and 0 otherwise, we have:

$$\begin{aligned}\frac{\partial F_{out}}{\partial \boldsymbol{A}_{ij}} &= 2\Big(\frac{\partial \boldsymbol{Y}(\boldsymbol{A})}{\partial \boldsymbol{A}_{ij}}\Big)^\top \boldsymbol{S}_{out}(\boldsymbol{Y}(\boldsymbol{A}) - \boldsymbol{Y}_{obs}) \\ &= 2\sum_{r=0}^{\infty}\Big(\frac{\partial \boldsymbol{T}_r}{\partial \boldsymbol{A}_{ij}}\Big)^\top \boldsymbol{S}_{out}(\boldsymbol{Y}(\boldsymbol{A}) - \boldsymbol{Y}_{obs}) \ .\end{aligned}$$

Using a triangular inequality, the bound on $\|\boldsymbol{Y}(\boldsymbol{A})\|$, and that $\|\boldsymbol{S}_{out}\boldsymbol{Y}_{obs}\| \leq \sqrt{|V_{out}|}y_\infty$ we get:

$$\|\boldsymbol{S}_{out}(\boldsymbol{Y}(\boldsymbol{A}) - \boldsymbol{Y}_{obs})\| \leq \frac{1 + \mu_{\min}\sqrt{|V_{tr}||V_{out}|}}{\mu_{\min}\sqrt{|V_{tr}|}}y_\infty \ .$$

By incorporating the resulting inequality in bounding the hypergradient, and by noticing that $\boldsymbol{S}_{out} = \boldsymbol{S}_{out}^2$ we have:

$$\left|\frac{\partial F_{out}}{\partial \boldsymbol{A}_{ij}}\right| \lesssim \frac{1+\mu_{\min}\sqrt{|V_{tr}||V_{out}|}}{\mu_{\min}\sqrt{|V_{tr}|}} y_\infty \sum_{r=0}^{\infty} \|\boldsymbol{S}_{out}\frac{\partial \boldsymbol{T}_r}{\partial \boldsymbol{A}_{ij}}\|$$
$$\lesssim \frac{1+\mu_{\min}\sqrt{|V_{tr}||V_{out}|}}{\mu_{\min}\sqrt{|V_{tr}|}} y_\infty \sum_{r=0}^{\infty} \left(\sum_{u\in V_{out}} \left|\frac{\partial (\boldsymbol{T}_r)_u}{\partial \boldsymbol{A}_{ij}}\right|^2\right)^{\frac{1}{2}} .$$

Using Lemma 4.6.3 and the hypotheses on i and j, for u in V_{out}, the term $\left|\frac{\partial (\boldsymbol{T}_r)_u}{\partial \boldsymbol{A}_{ij}}\right|$ is 0 if $r < q+k+1$, and bounded by $\frac{4\lambda y_\infty}{|E|\mu_{\max}^2\sqrt{|V_{tr}|}}(r-q-k)\nu^{r-1}$ otherwise. Hence:

$$\left|\frac{\partial F_{out}}{\partial \boldsymbol{A}_{ij}}\right| \lesssim \lambda \frac{\sqrt{|V_{out}|}+\mu_{\min}\sqrt{|V_{tr}||V_{out}|}}{\mu_{\min}|V_{tr}||E|\mu_{\max}^2} y_\infty^2$$
$$\times \sum_{r=q+k+1}^{\infty} (r-q-k)\nu^{r-1} .$$

Then we see that for $\nu < 1$ we have

$$\sum_{r=q+k+1}^{\infty} (r-q-k)\nu^{r-1} = \nu^{q+k}\sum_{r=1}^{\infty} r\nu^{r-1} ,$$

and $\sum_{r=1}^{\infty} r\nu^{r-1} = \frac{1}{(1-\nu)^2} = \frac{1}{\mu^2}$, which concludes the proof.

4.7 Alleviating hypergradient scarcity

In this section, we review strategies to mitigate the hypergradient scarcity problem. However, it is important that we make a distinction between resolving this issue and resolving the overfitting problem. Indeed, if gradient scarcity is also caused by the limited quantity of available labeled data, it is important to avoid confusion with traditional overfitting. In particular, while traditional overfitting is generally reduced by adding more training data, *gradient scarcity is still observed when optimizing edges far from labeled nodes regardless of the dataset size and the number of labels.* We study several strategies to mitigate hypergradient scarcity in the bilevel setting, but we emphasize that they might not lead to a better generalization error altogether.

Generalized edge refinement by optimizing $\boldsymbol{A}_{obs}^r$. As hypergradient scarcity is observed on edges connecting nodes distant from the labeled ones, a natural fix

is to reduce this distance. One way to do that is by refining edges in a power of $\boldsymbol{A}_{obs}$, as the matrix $\boldsymbol{A}_{obs}^r$ includes r-edge long connections between nodes. In our experiments we adopt $\boldsymbol{A}_{obs}^6$ as this notably expands the graph but does not achieve the extreme case where the result is a complete graph.

Graph regularization. Graph regularization is used to impose a prior structure on the learned graph, by adding a regularization term to F_{out} to penalize graphs with undesirable properties. For instance, Kalofolias (2016) propose the regularization term $-\gamma \mathbf{1}_n^\top \log \boldsymbol{A}\mathbf{1}_n$ for some $\gamma > 0$, to penalize low-degree nodes. We use this choice in the experiments, but note that imposing task-related priors and regularization terms could lead to better performance. This will be the topic of future work.

G2G for edge refinement. The third fix we suggest is metric learning using G2G models. In the outer problem, we propose to replace optimizing edge weights by optimizing the parameters of a G2G model to predict similarity between nodes. Let Θ be the weights of this model, and $\boldsymbol{A}_\Theta$ be its output graph, the G2G model we adopt is $(\boldsymbol{A}_\Theta)_{i,j} = \alpha\big((\boldsymbol{X}_i - \boldsymbol{X}_j)^2\big)$, where the square function is applied entrywise, $\alpha : \mathbb{R}^p \to \mathbb{R}$ is a Multi-Layer Perceptron (MLP) model consisting of k_{G2G} layers, each is of the form:

$$\boldsymbol{X}^{[l]} = \phi^{[l]}(\boldsymbol{X}^{[l-1]}\boldsymbol{W}_1^{[l]} + \mathbf{1}_n(\boldsymbol{b}^{[l]})^\top) \ ,$$

where $\boldsymbol{W}_1^{[l]} \in \mathbb{R}^{d_{l-1} \times d_l}, \boldsymbol{b}^{[l]} \in \mathbb{R}^{d_l}$ are learnable parameters, and d_l is the output dimensionality of the l-th layer. The parameters are gathered as follows $\Theta = \{\boldsymbol{W}_1^{[l]}, \boldsymbol{b}^{[l]}\}_{l=1}^{k_{G2G}}$.

4.8 Experiments

We[1] use two synthetic datasets, the first one, called synthetic dataset 1, is designed to examine hypergradient scarcity in the Laplacian regularization scenario. The second one is a binary classification dataset that can be used for both graph-based models. Due to the paradigm behind construction, we call it the cheaters dataset. We also illustrate our findings on the real-world Cora dataset.

Bilevel optimization routine: likewise the scenario of learning G2G models via bilevel optimization in Chapter 3, the problem Eq. (4.1) is intractable as neither the solution of the inner problem nor its gradient *w.r.t.* $\boldsymbol{A}$ (or to Θ with G2G models) has a closed form expression that can be evaluated. To overcome this

[1]Our *Python* implementation is available at https://github.com/hashemghanem/Gradients_scarcity_graph_learning.

difficulty, we employ ITD to approximate hypergradients. That is, we unroll (Gregor and LeCun, 2010) τ_{in} iterations of the gradient-based inner optimizer, then use the `Higher` package (Grefenstette et al., 2019) to trace iterations and perform higher-order automatic differentiation to obtain hypergradients. For both the inner and the outer optimizers, we consider the Adam algorithm (Kingma and Ba, 2014).

Synthetic dataset 1: we sample *i.i.d.* latent variables $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ for nodes uniformly at random from $[0, 1]$ with $n = 1536, p = 2$. The ground-truth graph $\boldsymbol{A}_{GT}$ is constructed s.t. $(\boldsymbol{A}_{GT})_{i,j} = 1$ if $\|\boldsymbol{X}_i - \boldsymbol{X}_j\|_2 < \sigma$, and 0 otherwise. σ is set to 0.06 in our experiments. Two distinct procedures were employed to sample the nodes that comprise V_{tr}, leading to two distinct realizations of the dataset as illustrated in Fig. 4.1(top). The first procedure randomly samples 100 nodes from the set V, hence V_{tr} is well-spread, whereas the second procedure selects the 100 nodes with the smallest Euclidean distance to the point $(0.5, 0.5)$, thus V_{tr} is concentrated in a small neighborhood in this case. In both cases, we randomly sample 25 nodes from V to construct V_{out}. The remaining nodes are equally divided between the validation and the test sets. Then, each node i in V_{tr} is labeled as follows:

$$(\boldsymbol{Y}_{obs})_i = \zeta(e^{-\frac{\|\boldsymbol{X}_i - \boldsymbol{a}_1\|^2}{2(0.2)^2}} + e^{\frac{-\|\boldsymbol{X}_i - \boldsymbol{a}_2\|^2}{2(0.2)^2}} + e^{\frac{-\|\boldsymbol{X}_i - \boldsymbol{a}_3\|^2}{2(0.2)^2}}) \ ,$$

where $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$ are randomly sampled from $[0, 1]^2$, and ζ is a scaling factor such that labels lie in $[0, 1]$. By this construction, the assumption of label smoothness on the graph is met, and the Laplacian regularization can be applied as in Eq. (1.1).

To generate labels for other nodes, we plug the labels of V_{tr} and $\boldsymbol{A}_{GT}$ in Eq. (1.1) with the Laplacian regularization model (1.2) fixing $\lambda = 1$, such that the solution holds the sought-for labels. This way, the ground-truth graph actually plays a role in labeling nodes in V_{out} and in the validation set.

The noisy observed graph is built upon random weights

$$(\boldsymbol{A}_{obs})_{i,j} = \xi_{i,j}(\boldsymbol{A}_{GT})_{i,j} \quad \text{where} \quad \xi_{i,j} \sim \mathcal{U}([0, 1]) \ .$$

Experiments on this dataset are done with the Laplacian regularization in the inner problem as in Eq. (1.1).

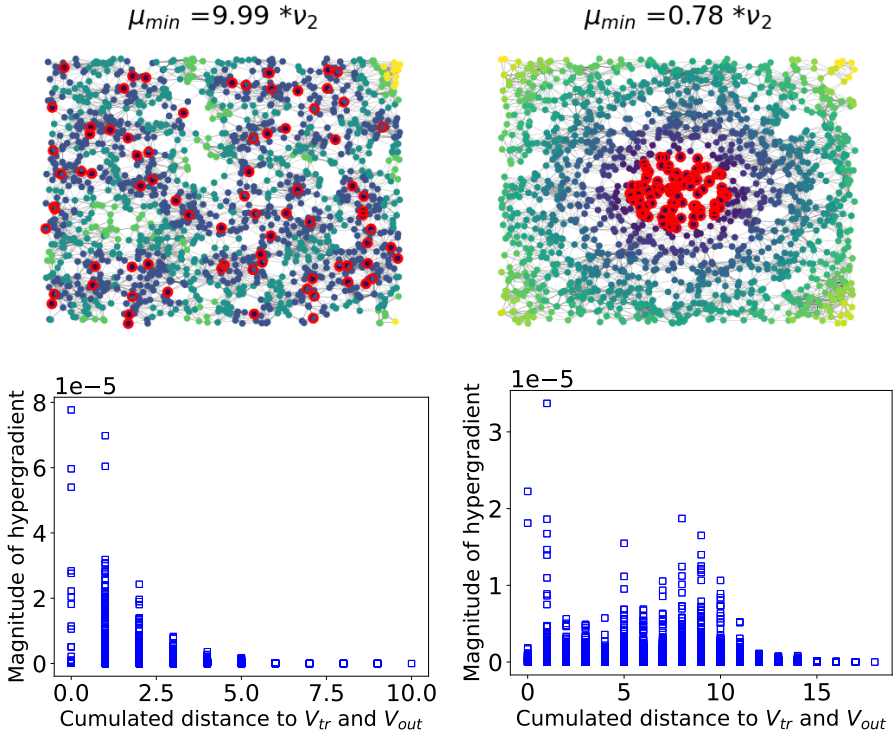


Figure 4.1 – hypergradient scarcity under the bilevel optimization setting on the synthetic dataset 1, adopting the Laplacian regularization in the inner problem. **Top**: illustration of the graph. The training nodes V_{tr} are circled in red, the colors correspond to the distance to V_{tr}. The eigenvalue $\mu_{\min}$ is given as a ratio of the smallest positive eigenvalue of $\tilde{\boldsymbol{L}}$. V_{out} is randomly sampled from V but not shown here. **Bottom**: Hypergradient magnitude $\left|\frac{\partial F_{out}}{\partial \boldsymbol{A}_{ij}}\right|$ with respect to the *sum* of distances to V_{tr} and V_{out}. **Left**: the training set V_{tr} is well-spread thereby aligned with the high-frequency eigenvectors of the graph, resulting in a *high* $\mu_{\min}$. The decrease of the hypergradients is sharp with the distance. **Right**: V_{tr} is aligned with the low-frequency eigenvectors of the graph, resulting in a *low* $\mu_{\min}$. The decrease of hypergradients magnitude is not as sharp as the previous case.

Cheaters dataset: nodes in this graph represent students in an exam classroom. Setting $n = 256, p = 10$, the *i.i.d.* features $\boldsymbol{X} \in \mathbb{R}^{256\times 10}$ are sampled uniformly at random from $[0, 1]$. For a node i, $\boldsymbol{X}_{i,0}$ represents the position of the according student in the classroom. For visualization purposes we enumerate nodes following the ascending order of $\boldsymbol{X}_{:,0}$. The remaining 9 features of a student represent the grades he is capable of scoring in the corresponding exam question. However, students tend to cheat with their neighbors in the graph. The ground-truth graph

$\boldsymbol{A}_{GT}$ is constructed as follows:

$$(\boldsymbol{A}_{GT})_{i,j} = \exp\left(-\|\boldsymbol{X}_{i,0} - \boldsymbol{X}_{j,0}\|_2^2 / 2\sigma^2\right) \ .$$

The observed graph $\boldsymbol{A}_{obs}$ is drawn from a random model as

$$(\boldsymbol{A}_{obs})_{i,j} \sim \text{Ber}\left((\boldsymbol{A}_{GT})_{i,j}\right).$$

We set $\sigma = 0.027$ s.t. the number of edges in $\boldsymbol{A}_{obs}$ approximates $n \log n$. Students cheat such that their grades $\boldsymbol{Y}_{grade}$ after the exam are

$$\boldsymbol{Y}_{grade} = \boldsymbol{A}_{GT}\boldsymbol{X}_{:,1:9}\mathbf{1}_9 \ .$$

A student passes the exam if his grade is greater than a threshold τ, *i.e.,* $(\boldsymbol{Y}_{obs})_i = 1$ if $(\boldsymbol{Y}_{grade})_i > \tau$ and 0 otherwise. We put $\tau = 60$ so that approximately half of students pass the exam. V_{tr} includes nodes in $\{0, 1, \ldots, n/8\} \cup \{7n/8, \ldots, n-1\}$, *i.e.,* near the two ends of the 1-dimensional class. $V_{out} = \{3n/8, \ldots, 5n/8\}$, *i.e.,* centered around the middle of the class. Remaining nodes are equally divided into a validation and a test set. Experiments on this dataset are done with a GNN model.

Real-world dataset: we validate our findings on the Cora dataset (Lu and Getoor, 2003). Cora is a citation datasets, where nodes represent research publications described by a bag of words, and edges stand for citations. The task is to classify articles *w.r.t.* their topic. We limit our experimentation on real-world datasets to the Cora dataset, as our empirical results are intended to establish a proof-of-concept. Therefore, we refrain from conducting experiments on other benchmark datasets.

Models: G2G and GNN models are implemented using *PyTorch* (Paszke et al., 2019) and *PyTorch* Geometric (Fey and Lenssen, 2019), respectively. The function α in the G2G model is an MLP with 2 hidden layers, each is followed by the *ReLu* activation function and has 16 neurons for the cheaters dataset and 32 neurons for Cora. The GNN has 1 hidden layer of 8 neurons for the cheaters dataset and 128 for Cora. This layer is followed *ReLu*, while the output is followed by the $softmax$ function.

Setup: we use Adam as the inner and the outer optimizer with the default parameters of *PyTorch*, except for the inner learning rate η_{in} and the outer one η_{out}, which are tuned from the set $\{10^{-4}, 10^{-3}, \ldots, 10\}$. The best values were $\eta_{in} = 10^{-2}$ with GNNs as a graph-based model, $\eta_{in} = 10^{-1}$ and $\eta_{in} = 10$ with the Laplacian regularization on Cora and on the synthetic dataset 1, respectively. On the cheaters dataset, $\eta_{out} = 10^{-3}$ adopting a G2G model, while $\eta_{out} = 10^{-2}$ in other cases. On the synthetic dataset 1, $\eta_{out} = 10^{-1}$. On Cora, $\eta_{out} = 10^{-2}$ in

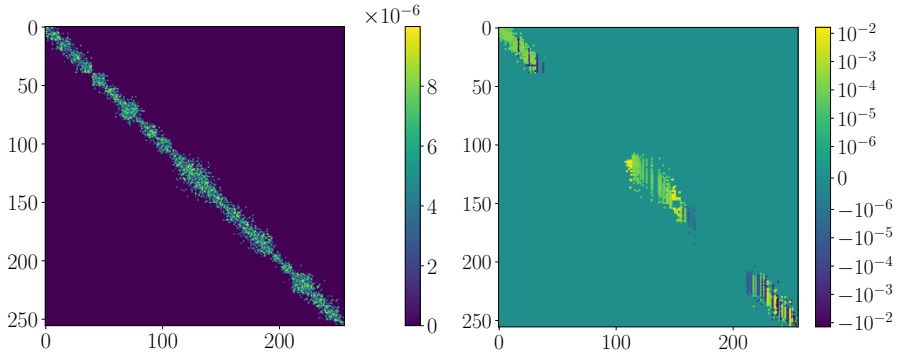


Figure 4.2 – Hypergradient scarcity observed when solving the edge refinement task with the bilevel optimization framework. We run the experiment on the cheaters dataset, and use a 2-layer GNN as a graph-based model. Left: graph initialization. Right: hypergradient at an arbitrary outer iteration, namely 9. It is clear that the hypergradient on edges between unlabeled nodes far from the ones in $V_{out} \cup V_{tr}$ equals zero. Recall that $V_{tr} = \{0, 1, \ldots, 32\} \cup \{224, \ldots, 255\}$ and $V_{out} = \{96, \ldots, 160\}$.

all experiments without a G2G model, otherwise $\eta_{out} = 10^{-4}$ adopting the GNN model, and $\eta_{out} = 10^{-3}$ adopting the Laplacian regularization. We set, with a grid search, τ_{in} to 200 for the cheaters dataset, 500 for the synthetic dataset 1 and Cora adopting the Laplacian regularization, and 100 for Cora with a GNN model. In experiments on the cheaters dataset, we multiply the default initialization of the last layer of the G2G model by 10^{-5} s.t. its output edges at the first iteration are of small magnitude. We adopt this strategy to measure the level of scarcity by counting the number of learned edges of magnitude greater than a chosen threshold. GNN weights $\boldsymbol{W}$ and the initialization of labels when using the Laplacian regularization are initialized at random after each outer iteration, using Xavier initialization and uniformly at random from $[0, 1]$, respectively. Edges to be refined are initialized uniformly at random from $[0, 1]$, except for experiments on the cheaters dataset where the interval becomes $10^{-5} * [0, 1]$. We set the number of outer iterations τ_{out} to 150 while ensuring convergence, and we select the graph (or the G2G weights) with the highest validation accuracy. We set $\lambda = 1$ in training when considering the Laplacian regularization, as we expect the bilevel algorithm to learn this parameter by scaling the learned adjacency matrix. When applying the Laplacian regularization fed with $\boldsymbol{A}_{obs}$ on Cora, we set $\lambda = 0.1$ after a grid search. γ in the graph regularization term is set to 1 following a grid search on the set $\{10^{-3}, 10^{-2}, \ldots, 10\}$.

4.8.1 Hypergradient scarcity with GNN models

In this experiment, we consider a 2-layer GNN model in the bilevel framework. We solve the edge refinement task (4.1) on the cheaters dataset, where ℓ in Eqs. (1.1) and (4.1) is the CCE function. Fig. 4.2(left) depicts the initialization of the adjacency matrix. It also shows what edges are to be optimized, that is, edges whose initialization is nonzero. In Fig. 4.2(right), we show the hypergradient at the outer iteration 9, which is arbitrarily chosen, where it is clear that edges between unlabeled nodes far from the ones in the union $V_{out} \cup V_{tr}$ get no supervision during the training process. Recall that $V_{tr} = \{0, 1, \ldots, 32\} \cup \{224, \ldots, 255\}$ and $V_{out} = \{96, \ldots, 160\}$. This aligns with our findings, which state that edges between nodes at least 2-hop from nodes in $V_{out} \cup V_{tr}$ receive zero hypergradients. This, as seen in Fig. 4.3, leads to a learned graph that overfits training nodes and even generalizes worse than $\boldsymbol{A}_{obs}$. We believe that this is the main reason why the method proposed in Wan and Kokel (2021), which solves a graph sparsification task using bilevel optimization with a GNN classifier, does not necessarily improve over the observed graph.

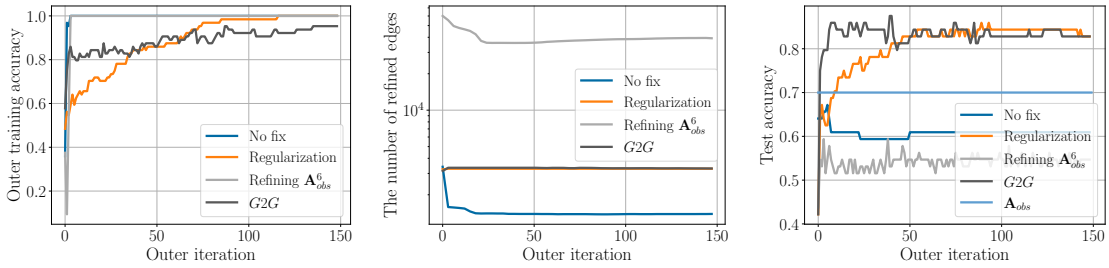
4.8.2 Hypergradient scarcity with Laplacian regularization

We here examine hypergradient scarcity when adopting the Laplacian regularization in the inner problem. We run the bilevel optimizer to solve the edge refinement task on the synthetic dataset 1. The dataset corresponds to a regression problem, so ℓ in Eqs. (1.1) and (4.1) is the MSE loss function.

In Fig. 4.1(bottom), we plot the absolute value of hypergradients at the outer iteration 6 as a function of the edge cumulative distance to V_{tr} and V_{out}, which is defined as follows: we compute $q + k$, the sum of hop distances to V_{tr} and V_{out}, respectively, for its both endpoint nodes, then we take the minimum of the two results. One observes the hypergradient scarcity phenomenon, since hypergradients decay exponentially as the edge distance increases. This validates our analysis articulated in Theorem 4.6.2. In addition, we observe in practice that μ is nevertheless quite small, and that our bound in Theorem 4.6.2 is quite loose. Another observation is that the decrease rate is higher when V_{tr} is well-spread in the graph. Deriving a tighter bound on the magnitude of hypergradients and investigating the link between the distribution of labeled nodes and this bound will be the subject of a future work.

4.8.3 Testing solutions to mitigate hypergradient scarcity

We run our experiments on the cheaters dataset using the 2-layer GNN as a graph-based model. In each experiment, we run our bilevel optimization framework with



(a) (b) (c)

Figure 4.3 – Efficiency of proposed solutions to hypergradient scarcity *w.r.t.* the number of refined edges and the generalization capacity. An edge is considered well refined if its learned weight is larger than one percent of the maximum learned edge weight. The solutions are graph regularization with $-\mathbf{1}^\top \log \boldsymbol{A}\mathbf{1}$, metric learning using a G2G model, and generalized edge refinement by refining edges in $\boldsymbol{A}^6_{obs}$. (a): training accuracy on V_{out}. (b): number of refined edges. (c) test accuracy.

one of the suggested fixes. We consider two criteria to measure the efficiency of each solution, the first one is counting the number of refined edges. At any outer iteration, we say that an edge is refined if its learned weight is greater that one percent of the maximum learned edge weight at the same iteration. Recall that we initialize the graph/G2G with small weights ($\approx 10^{-5}$). The second criterion is the test accuracy. The first criterion assesses the ability to alleviate hypergradient scarcity, while the second assesses the generalization to unseen nodes during training, and thus if the learned graph is meaningful.

Fig. 4.3 shows that all three fixes produce better results *w.r.t.* the first criterion, as the number of refined edges is larger at almost every iteration, with optimizing edges in $\boldsymbol{A}^6_{obs}$ being the most efficient, and the G2G model and graph regularization having similar performance. Moreover, one notices that this number decreases with the iteration when refining edges in $\boldsymbol{A}_{obs}$ or in $\boldsymbol{A}^6_{obs}$, which is expected as only a small portion of edges receive supervision; however this portion is larger when refining $\boldsymbol{A}^6_{obs}$.

Regarding the second criterion, the G2G model and the graph regularization generalize well, as both combat hypergradient scarcity without increasing (or even by decreasing) the number of parameters to learn. On the other hand, optimizing edges in $\boldsymbol{A}^6_{obs}$ deteriorates performance in the test phase. A likely explanation is that by expanding the graph, we increase the number of parameters to learn, which means a more complex model that is more likely to overfit training nodes. This experiment illustrates that **hypergradient scarcity is not the traditional overfitting** related to data/label scarcity, and resolving it does not necessarily

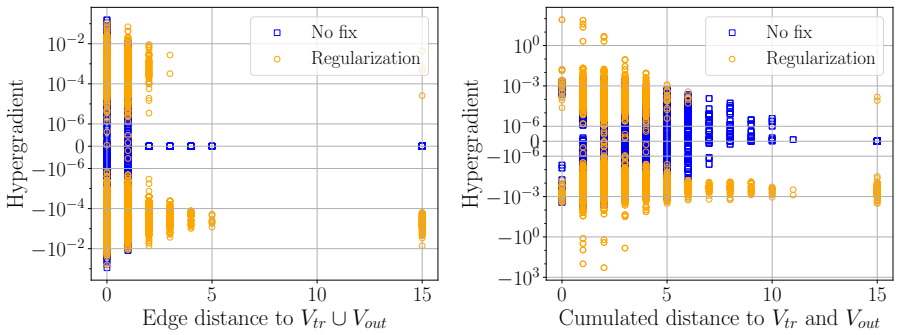


Figure 4.4 – Observing hypergradient scarcity and the effect of graph regularization on Cora. Left: adopting the GNN as the graph-based method. Right: adopting the Laplacian regularization model. We plot the hypergradient against edge distance. In connected components without at least a node from each of V_{tr} and V_{out} in the Laplacian regularization case (or without a node from $V_{tr} \cup V_{out}$ in the GNN case), edge distance is not defined. We assign the distance 15 to edges in such components for visualization purpose.

promote better generalization.

4.8.4 Results on Cora

We use bilevel optimization (4.1) to solve an edge refinement task on Cora, trying both the GNN and the Laplacian models. Here the downstream task is a multi-label classification problem and ℓ is the CCE function. We depict in Fig. 4.4 the received hypergradient on edges at outer iteration 9 as a function of their distance to labeled nodes. For the Laplacian regularization case, that is the edge cumulative distance to V_{tr} and V_{out} as defined in Section 4.8.2. To compute the edge distance in the GNN case, we compute for each of its endpoint nodes its hop distance to $V_{tr} \cup V_{out}$, then we take the minimum. In accordance with our analysis, the figure displays a null hypergradient for distances greater than 2 in the GNN case, while the Laplacian regularization scenario exhibits a hypergradient that diminishes exponentially with distance.

Regarding the generalization capacity, Table 4.1 shows that the learned graph is inferior to $\boldsymbol{A}_{obs}$ in the GNN case for the test error. Given that the learned graph achieves $100\%, 94.9\%$ accuracies on V_{tr}, V_{out}, respectively, one concludes that hypergradient scarcity provokes overfitting. This is indeed expected due to the extreme scarcity in the GNN scenario, as edges of distance greater than 2 keep their random initialization after the training process. This is, however, not the case in the Laplacian regularization scenario as most edges are of distance less

Table 4.1 – Accuracies obtained on Cora when the graph-based model is trained using the output graph of the Bilevel Optimization (BO) framework, the same framework equipped with graph regularization, the same framework optimizing a G2G model as in Chapter 3. We also benchmark against GAM (the result is reported from the according paper) and against $\boldsymbol{A}_{obs}$. For each graph-based method, we report test accuracy in the according first line and training accuracy on V_{out} in the second one. Training accuracy on V_{tr} equals 100% for all methods.

Graph	$\boldsymbol{A}_{obs}$	BO	BO+regularization	BO+G2G	GAM
GNN	77.0	76.2	80.3	82.0	84.8
	77.4	94.9	94.1	97.4	-
Laplacian	71.7	76.2	78.3	76.2	-
	71.0	81.9	83.2	83.5	-

than 11, thereby they do not exhibit damped hypergradients and the impact on generalization is not observed.

Next, we test the efficiency of the proposed solutions to mitigate hypergradient scarcity. We do not try learning a power of $\boldsymbol{A}_{obs}$ as the memory requirement goes beyond the limits we have access to. Results in Fig. 4.4 prove the efficiency of graph regularization as all edges receive non-zero hypergradients with a comparable magnitude to those on edges of small distance. Note that hypergradients are received on the G2G weights when it is deployed, not on edges, so we do not depict them in this figure. Regarding the impact on generalization, Table 4.1 shows that both fixes yield significant improvements in test accuracy over $\boldsymbol{A}_{obs}$ with the GNN model. In the Laplacian regularization case, graph regularization produces a higher test accuracy, unlike the G2G model which generalizes equally good as when learning directly edge weights. We also notice that GNN model leads to superior results in all scenarios, with a notable gap for the G2G model and graph regularization, and when directly using $\boldsymbol{A}_{obs}$. This is expected, as the Laplacian regularization promotes similarity between connected nodes but, unlike GNNs, is not a supervised-based method. We finally point out that the bilevel optimization framework with either fix does not achieve state-of-the-art results produced by GAM with the same GNN model.

Other experiments suggest that although G2G models alleviate hypergradient scarcity, regardless of the number of neurons in its layers, the generalization performance is sensitive to this number and if set large, clear overfitting is observed.

Chapter 5

Conclusion

The main focus of this thesis was graph learning with bilevel optimization for Semi-Supervised Learning (SSL) tasks. The first part of it was dedicated to learning analysis-sparsity priors for denoising problems (Chapter 2). This problem is indeed a graph learning problem in graph total variation-related tasks. Since the problem is intractable and hypergradients are not accessible, we proposed to employ ITerative Differentiation (ITD) to approximate hypergradients *w.r.t.* priors and then learn them using a gradient-descent optimizer. Despite non-smoothness present due to the ℓ_1 norm in the inner problem, and that no relaxation regime is adopted in our method, experiments proved its effectiveness in extracting the analysis-sparsity operator from 1D piecewise constant signals and from 2D images. Moreover, we proposed a simple column-wise centering projection in the former application, and empirically proved that it increases stability and extracts operators with higher quality than a representative of relaxation-based methods.

The second part of this thesis was concerned with real-world graphs being noisy or not given in SSL applications. We proposed a bilevel optimization framework to train parametric models, which we referred to as G2G, on predicting node similarity thereby better graphs (Chapter 3). G2G models have lower memory and training costs compared to optimizing edge weights thanks to its shared parameters, and have the advantage to generalize by expanding the graph on new points. Similarly, we resorted to ITD to approximate hypergradients *w.r.t.* G2G weights, and empirically showed that unrolling only the last 10 inner iterations produces high-quality graphs, which notably saves memory and training time. Experiments on synthetic data proved the capacity of our method to learn graphs in the absence of an observed graph, *i.e.,* learning the similarity criterion from features. Compared to observed graphs, experiments on real datasets showed that G2G boosts the performance of GNN models and generalizes well on the test set.

Next, we identified and studied the hypergradient scarcity phenomenon when directly optimizing observed edges instead of G2G models (Chapter 4). This phenomenon manifests as a situation where edges linking nodes far from labeled nodes receive zero gradients. We proved that this problem occurs for GNNs as the graph-based model. We also proved that replacing GNNs by the Laplacian regularization does not resolve the issue; however, the phenomenon is less severe: we bounded the magnitude of hypergradients and proved that they are exponentially damped with distance to labelled nodes. To alleviate this issue, we proposed to resort to metric learning using G2G models as in Chapter 3, graph regularization, and refining edges in a power of the observed adjacency matrix. Our experiments validated our findings, and privileged the first two solutions over the latter. Moreover, we showed that alleviating the hypergradient scarcity does not necessarily alleviate overfitting.

Many questions remain unanswered and are left for future works.

Learning analysis-sparsity operators of unknown dimensions. In Chapter 2, our framework assumed that the number of columns in the analysis-sparsity operator is given. Recall that this number in applications related to graph total variation is the number of edges in the graph. Learning this hyperparameter when not given seems to be challenging in this point, since it is a discrete variable therefore cannot be learned using gradient-based methods and since grid search methods can be expansive due the large number of possible values.

The number of parameters in G2G models. At the end of Chapter 4 where G2G models are trained to solve *edge refinement* tasks, we highlighted that the number of parameters in G2G models affects their generalization performance and that large values cause overfitting. Interestingly with *graph construction* tasks in Chapter 3, specifically in Section 3.5.1, the G2G managed to recover the ground-truth graph while having a relatively large number of neurons, even though no observed graph is given in this setting. This showed an information-preservation phenomenon when constructing the latent position graph and its labels. Investigating the factors that promote this phenomenon and how the edge refinement setting affects it is an important question and a good continuation of our work.

Hypergradient scarcity in the Laplacian regularization case. In Chapter 4, we proved that the hypergradient decreases exponentially with distance to labeled nodes in the Laplacian regularization case. However, experiments in Section 4.8.2 showed that the empirical rate is much faster than the theoretical one hence implying that our bound is loose. We also observed that the decrease rate is higher when the inner training set V_{tr} is well-spread through the graph as shown

in Fig. 4.1. Deriving a tighter bound on the magnitude of hypergradients and investigating the link to the distribution of labelled nodes is an interesting point to consider. This could be achieved by considering the zero pattern in $\boldsymbol{B} = \frac{\boldsymbol{S}_{in}}{|V_{tr}|} + \lambda \frac{\boldsymbol{L}}{|E|}$ in Lemma 4.6.3, rather than treating it as a generic matrix as we did. Alternatively, employing different techniques than the Neumann expansion can also contribute to the analysis. Last, our analysis focused only on regression tasks, we believe that following similar steps can extend it to the classification setting.

Appendix A

Automatic Differentiation (AD)

In this appendix, we introduce Automatic Differentiation (AD), which is a key ingredient in the ITD bilevel optimization routine. We first present the different techniques used to evaluate derivatives, including AD, symbolic and numerical differentiation, while highlighting AD's capacity to differentiate iterative algorithms. We also present the two implementation schemes of AD, the *forward mode* and the *reverse mode* adopted in this work. We finally present our empirical contribution, which points out the limitation of AD in differentiating a high number of iterations depending on the choice of the iterative optimizer.

Gradients evaluation is a key stage in optimization and machine learning, where one usually implements a computer program to maximize (or minimize) a target function moving along (or opposite to) its gradient. Four methods exist to compute gradients, the first method includes manually writing down the analytical gradient, and letting a program evaluate it. For complex functions, this can be time consuming, extremely difficult, and prone to mistakes. Moreover, this technique works on functions with a closed-form expression that can be evaluated, which is not the case in the problems of interest in this work. The second method approximates gradients using numerical differentiation, which is easy to implement, but is exposed to round-off errors (Jerrell, 1997), and still expensive in case of a high number of variables.

The remaining two methods have a notable intersection, algorithmically speaking. Symbolic differentiation, as the third method, is automatically performed by computer tools like Mathematica, which output a symbolic expression of derivatives. In detail, after the *expression* of a function is written, such tools break it apart into a sequence of basic operations (sum, product, composition, etc..) applied on functions whose derivative is known (Grabmeier and Kaltofen, 2003).

Then, the overall derivative is made up by using the according differentiation rules, *e.g.*, $\frac{d}{dx}(f+g) = \frac{df}{dx} + \frac{dg}{dx}$, also the chain rule for function composition. This derivative then can be evaluated at any given point.

Symbolic differentiation addresses the problem of errors, and the large time needed to manually calculate derivatives. On the other hand, incautious application of the previous rules can lead to exponential growth in the derivative expression, as some terms are unnecessarily repeated resulting in a higher evaluation cost, this is known as expression swell (Corliss, 1988). More importantly, symbolic differentiation works only on functions of closed-form expression, which severely limits the expressivity of models that can be differentiated.

The fourth method is *Automatic Differentiation* (AD), that is of interest in this work as it mitigates previous drawbacks. AD manipulates the computation flow in a computer program (not symbolic expressions). The underlying claim is: all numerical computations can be reduced to compositions of elementary operations, whose derivative rule is known (Verma, 2000).

AD technique works as follows (Baydin et al., 2018), first a value is assigned to each input variable, then during the execution of the program: *i)* AD traces all new defined variables that are dependent on the ones we want to differentiate for; *ii)* once an operation is performed on a dependent variable, say v_i, to evaluate another v_j, directly compute the derivative value dv_j/dv_i; *iii)* accumulate the derivatives in step 2 through the chain rule; this gives the derivative value (not expression) of the whole composition *w.r.t.* a chosen variable.

Although AD and symbolic differentiation rely on the chain rule, AD can efficiently differentiate not only closed-form formulas, but also iterative algorithms. That is to say, AD can compute the derivative of the output of an iterative algorithm *w.r.t.* its input variables. Moreover, AD does not require these algorithms to perform the same set of operations at each iteration, as it can trace the overall computation flow when the operations change from one iteration to the other, which can be the case when using conditioned statements and loops (*if, while, for*). This makes AD suitable for gradient-based optimization.

In practice, AD can be implemented in two ways: *forward mode* and *reverse mode*. We present both schemes in the following sections.

A.1 Forward mode

This mode specializes in computing the directional derivative. That is, given a vector that defines a direction in the input space, the forward mode computes the

Computation flow to evaluate $\boldsymbol{y}$

$$
\begin{array}{lll}
x_1 & & = 3 \\
x_2 & & = 1 \\
v_1 & = \ln x_1 & = 1.1 \\
v_2 & = x_1 x_2 & = 3 \\
v_3 & = \cos(v_2) & = \text{-}0.99 \\
y & = v_1 + v_3 & = 0.11
\end{array}
$$

Computation flow of AD forward mode

$$
\begin{array}{lll}
\dot{x}_1 & & = 1 \\
\dot{x}_2 & & = 0 \\
\dot{v}_1 & = \dot{x}_1 \frac{\partial v_1}{\partial x_1} & = 0.33 \\
\dot{v}_2 & = \dot{x}_1 \frac{\partial v_2}{\partial x_1} + \dot{x}_2 \frac{\partial v_2}{\partial x_2} & = 1 \\
\dot{v}_3 & = \dot{v}_2 \frac{\partial v_3}{\partial v_2} & = \text{-}0.14 \\
\dot{y} & = \dot{v}_1 \frac{\partial y}{\partial v_1} + \dot{v}_3 \frac{\partial y}{\partial v_3} & = 0.19
\end{array}
$$

Table A.1 – Illustration of the forward mode of AD, we consider the example $y = f(x_1, x_2) = \ln x_1 + \cos(x_1 x_2)$ at $\boldsymbol{x} = (x_1, x_2) = (3, 1)$. We compute the derivatives $\frac{\partial y}{\partial x_1}|_{x_1=3}$, thereby we initialize with $\dot{x}_1 = 1, \dot{x}_2 = 0$. Left: the computation trace of y, right: AD forward accumulation of derivatives using the chain rule.

derivative of each output component along this direction. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be the input variable, $\boldsymbol{v}$ be an intermediate variable that is defined through the program execution, and $\boldsymbol{y} = (y_1, \ldots, y_m)$ be the program output that is a function of $\boldsymbol{v}$, the forward scheme takes in input the direction vector $\dot{\boldsymbol{x}} = (\dot{x}_1, \ldots, \dot{x}_n)^\top$, and construct derivatives using the chain rule starting from $\boldsymbol{x}$, then $\boldsymbol{v}$, and ending in $\boldsymbol{y}$. Formally, it first evaluates $\dot{\boldsymbol{v}} = \boldsymbol{J}_{\boldsymbol{v}}(\boldsymbol{x})\dot{\boldsymbol{x}}$, then $\dot{\boldsymbol{y}} = \boldsymbol{J}_{\boldsymbol{y}}(\boldsymbol{v})\dot{\boldsymbol{v}}$. The output $\dot{\boldsymbol{y}}$ is nothing but the Jacobian-vector product $\boldsymbol{J}_{\boldsymbol{y}}(\boldsymbol{x})\dot{\boldsymbol{x}}$. Our example in Table A.1 shows this process step by step.

It is noteworthy that once an intermediate variable is evaluated in the program, it is directly augmented with the value of its derivative in the forward mode. On the one hand, this gives that the computational flow of derivatives has the same order as the main program, and the final derivatives are obtained once the output is evaluated. On the other hand, there is no need to store the computation flow that led to a variable if it is no further used in the program, *i.e.,* this mode has a low memory cost. One may also notice that to compute the derivatives *w.r.t.* each component i in the input variable, a different call of the forward mode is needed considering the direction $\dot{\boldsymbol{x}}$, where $\dot{x}_j = 1$ if $j = i$ and 0 otherwise.

A.2 Reverse mode

This mode returns the value of the partial derivative *w.r.t.* each component in the input variable. In contrast to the forward mode, here we first complete the computations needed to evaluate the output, then we start the ones needed to evaluate derivatives. That is, the chain rule accumulates derivatives in an opposite order to the computer program. Given an initialization vector $\bar{\boldsymbol{y}} = (\bar{y}_1, \ldots, \bar{y}_m)$,

Computation flow to evaluate $\boldsymbol{y}$ ↓

x_1		$= 3$
x_2		$= 1$
v_1	$= \ln x_1$	$= 1.1$
v_2	$= x_1 x_2$	$= 3$
v_3	$= \cos(v_2)$	$= -0.99$
y	$= v_1 + v_3$	$= 0.11$

Computation flow of AD reverse mode ↑

$\bar{x}_1$	$= \bar{v}_2 \frac{\partial v_2}{\partial x_1} + \bar{v}_1 \frac{\partial v_1}{\partial x_1}$	$= 0.19$
$\bar{x}_2$	$= \bar{v}_2 \frac{\partial v_2}{\partial x_2}$	$= -0.42$
$\bar{v}_1$	$= \bar{y} \frac{\partial y}{\partial v_1}$	$= 1$
$\bar{v}_2$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_2}$	$= -0.14$
$\bar{v}_3$	$= \bar{y} \frac{\partial y}{\partial v_3}$	$= 1$
$\bar{y}$		$= 1$

Table A.2 – Illustration of the reverse mode of AD, we consider the example $y = f(x_1, x_2) = \ln x_1 + \cos(x_1 x_2)$ at $\boldsymbol{x} = (x_1, x_2) = (3, 1)$. We compute the derivatives $\bar{x}_1 = \frac{\partial y}{\partial x_1}|_{x_1=3}$ and $\bar{x}_2 = \frac{\partial y}{\partial x_2}|_{x_2=1}$. Left: the computation trace of y, right: AD reverse accumulation of derivatives using the chain rule.

this mode computes derivatives starting from $\bar{\boldsymbol{v}} = \boldsymbol{J}_{\boldsymbol{y}}^{\top}(\boldsymbol{v})\bar{\boldsymbol{y}}$, then $\bar{\boldsymbol{x}} = \boldsymbol{J}_{\boldsymbol{v}}^{\top}(\boldsymbol{x})\bar{\boldsymbol{v}}$. In other words, $\bar{\boldsymbol{x}}$ is the transpose Jacobian-vector product $\boldsymbol{J}_{\boldsymbol{y}}^{\top}(\boldsymbol{x})\bar{\boldsymbol{y}}$. Our example in Table A.2 shows this process step by step.

Note that to compute the partial derivatives of y_i with respect to components in $\boldsymbol{x}$, $\bar{\boldsymbol{y}}$ must be defined as follows: $\bar{\boldsymbol{y}}_j = 1$ if $i = j$ and 0 otherwise. Therefore, one call of this mode is needed to compute the partial derivatives of each component in the output $\boldsymbol{y}$. Also note that the computation flow outputting $\boldsymbol{y}$ needs to be stored when deploying this mode, *i.e.,* the reverse mode has a higher memory cost.

A.3 Reverse mode vs. forward mode

As we previously mentioned, the number of calls in the forward (reverse) mode equals the input (output) dimensionality n (m). For that reason, the forward mode is usually preferred when $n \ll m$ due to its efficiency in that scenario. Conversely, the reverse mode is more suitable when $m \ll n$. Following this logic, we adopt the reverse mode in our work since the output is a scalar loss function and the input is of high dimensionality.

A.4 Sensitivity to the inner optimizer

In the section, we present one of our empirical observations which implies that AD, thereby the ITD method, fails to differentiate through a high number of iterations depending on the choice of the inner optimizer. This is, to the best of our knowledge, the first work that identify this issue in the literature.

Motivated by the popular optimizer called the Adaptive Moment Estimation algorithm (Adam) significantly outperforming SGD in all graph-based SSL experiments (Chapters 3 and 4), we decided to adopt it as the inner optimizer. Let $\boldsymbol{W}_t$ be the parameters of the inner problem at the t-th inner iteration, then Adam's update rule is given by (Kingma and Ba, 2014):

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta_{in}\widehat{\boldsymbol{m}}_t/(\sqrt{\widehat{\boldsymbol{v}}_t} + \epsilon) \ , \tag{A.1}$$

where $\widehat{\boldsymbol{m}}_t = \boldsymbol{m}_t/(1-\zeta_1^t)$, $\widehat{\boldsymbol{v}}_t = \boldsymbol{v}_t/(1-\zeta_2^t)$ are the bias-corrected moment estimates, ζ_1, ζ_2 are exponential decay rates defined by user (so is ϵ), $\boldsymbol{m}_t = \zeta_1\boldsymbol{m}_{t-1} + (1-\zeta_1)\nabla_{\boldsymbol{W}_{t-1}}F_{in}$, $\boldsymbol{v}_t = \zeta_2\boldsymbol{v}_{t-1} + (1-\zeta_2)(\nabla_{\boldsymbol{W}_{t-1}}F_{in})^2$ are the biased moment estimates with $\boldsymbol{m}_0 = \boldsymbol{v}_0 = \boldsymbol{0}$.

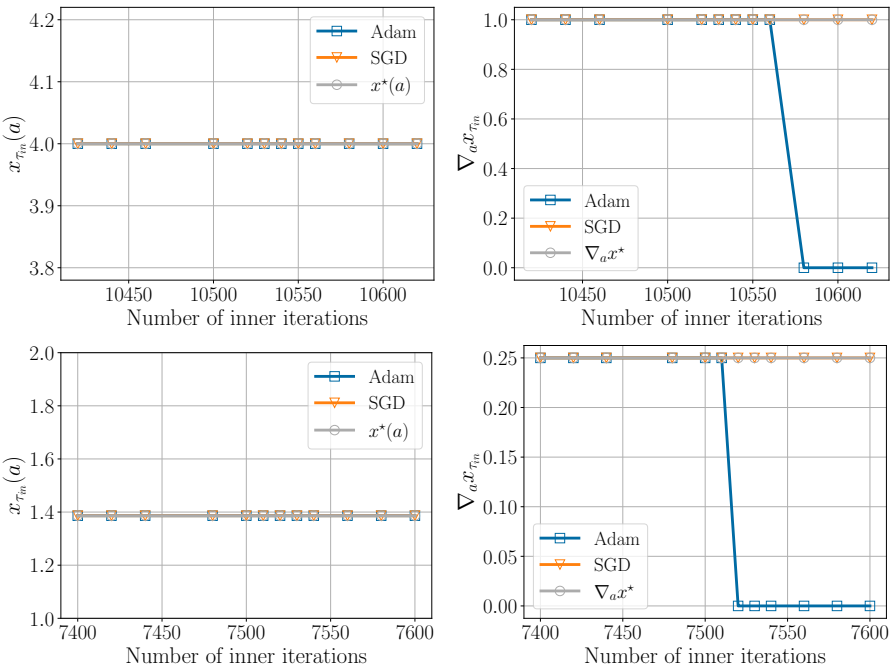


Figure A.1 – Examining the capacity of ITD in differentiating through a large number of inner iterations. We experiment on toy inner problems: Eq. (A.2) and Eq. (A.3). We also consider two inner optimizers: Adam and SGD. We set the outer level variable $a = 4$. For each pair (inner problem, optimizer), we vary the number of iterations τ_{in}, then we run AD to compute the gradient of its output *w.r.t.* a. We obtain the gradient $\nabla_x F_{in}$ using AD implemented in the `PyTorch` package (Paszke et al., 2019). To differentiate through the iterative optimizer and compute $\nabla_a x_{\tau_{in}}$, we use AD implemented in the `Higher` package(Grefenstette et al., 2019). **Top:** experiments on Eq. (A.2). **Down:** experiments on Eq. (A.3). **Left:** the optimizer output $x_{\tau_{in}}(a)$. **Right:** the gradient $\nabla_a x_{\tau_{in}}$. We observe that in all experiments, $x_{\tau_{in}}$ converges to the true solution $x^\star$. However, AD fails to differentiate through Adam's iterations when τ_{in} gets large, while it succeeds in the case of SGD.

To verify the proficiency of ITD in this case, we designed simple inner problems to examine the capacity of AD in differentiating through Adam's iterations and producing an accurate approximation of the hypergradient. We found that when the number of iteration is large, AD-based hypergradient either diminishes to zero, explodes or holds not-a-number "*NaN*" output. To illustrate this, let us consider

the following two toy inner problems:

$$x^\star(a) = \min_{x \in \mathbb{R}} F_{in}(x, a) = (x - \exp(a))^2 \ , \tag{A.2}$$

$$x^\star(a) = \min_{x \in \mathbb{R}} F_{in}(x, a) = (x - a)^2 \ , \tag{A.3}$$

where $a \in \mathbb{R}$ is the outer variable, *i.e.,* Θ in Eq. (1.7) is the set $\{a\}$. In each scenario, we set $a = 4$ (chosen arbitrarily) and solve the inner problem once using the Adam optimizer and a second time using the SGD optimizer, while varying the number of iterations τ_{in}. In all experiments we initialize x with the value 0, and the learning rate is set to 0.1. Fig. A.1 shows the optimizer output $x_{\tau_{in}}(a)$ and its gradient $\nabla_a x_{\tau_{in}}$ evaluated using AD. Although both optimizers converge to the true solution for every choice of τ_{in}, we observe that AD fails to differentiate through Adam's iterations when τ_{in} gets large (larger than 10555 for Eq. (A.2) and 7500 for Eq. (A.3)), while it succeeds in the case of SGD. The reason behind this behavior is still ambiguous. It is possibly related to the implementation of AD in the `Higher` package (Grefenstette et al., 2019), which we use to differentiate iterative optimizers. We leave the investigation of this issue for future work.

Fortunately, the number of iterations τ_{in} in our experiments has small values that do not exceed 500, which are set based on the validation loss while ensuring convergence too. To further ensure the stability of ITD in our experiments, we compared the produced hypergradient against the theoretical approximations of the true hypergradient whenever available. For instance, we saw in Section 1.7.3 that the solution of the inner problem when adopting the Laplacian regularization in regression SSL tasks has a formula that can be analytically approximated using truncated Neumann series. The hypergradient adopting this approximation in the inner problem can be analytically derived and evaluated in order to compare it against the one produced by ITD. We found that both hypergradients are close to each other. Last, we considered that the output of the bilevel optimization with Adam outperforming the one with SGD as another indication that ITD produces hypergradients of high quality. The performance criteria here are again the validation and outer losses.

Appendix B

Bilevel learning of analysis-sparsity priors

B.1 Solving the dual problem of Eq. (2.1b) with FISTA

Fortunately, the term $\iota_{B_\infty}(\boldsymbol{z})$ in the dual problem in Eq. (2.2) has a simple proximal function given by Π_{B_∞}: the orthogonal projection on the ball B_∞. So indeed, we solve Eq. (2.2) with an accelerated FB algorithm, namely the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) (Beck and Teboulle, 2009). At each iteration we update $\boldsymbol{z}$ as follows:

$$\begin{aligned} \boldsymbol{z}_{i+1} &= \operatorname{prox}_{\iota_{B_\infty}}\left(\boldsymbol{q}_i - \eta_{in}\nabla(\frac{1}{2}\left\|\boldsymbol{\Omega}\boldsymbol{q}_i - \boldsymbol{y}\right\|_2^2)\right) \\ &= \Pi_{B_\infty}\left(\boldsymbol{q}_i - \eta_{in}\boldsymbol{\Omega}^\top(\boldsymbol{\Omega}\boldsymbol{q}_i - \boldsymbol{y})\right) \end{aligned} \tag{B.1}$$

s.t. $\boldsymbol{q}_1 = \boldsymbol{z}_0$ is the initialization of $\boldsymbol{z}$, η_{in} is the step size, and:

$$\begin{aligned} \boldsymbol{q}_{i+1} &= \boldsymbol{z}_i + \frac{t_i - 1}{t_{i+1}}(\boldsymbol{z}_i - \boldsymbol{z}_{i-1}) \\ t_{i+1} &= \frac{1}{2}(1 + \sqrt{1 + 4t_i^2}) \;\; ; t_1 = 1. \end{aligned} \tag{B.2}$$

To conclude with the inner part: having $\boldsymbol{\Omega}$ and $\boldsymbol{y}$ as *input*, one compute $\hat{\boldsymbol{z}}(\boldsymbol{\Omega}, \boldsymbol{y})$ with a sufficient number of updates as in Eq. (B.1), then get $\hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y})$ *in output* using Eq. (2.3).

B.2 Proximal operator of the analysis-sparsity regularizer

We show in this section why evaluating the proximal operator of the analysis-sparsity regularizer $\|\boldsymbol{\Omega}^\top \cdot\|_1$ is not efficient, thus, as a straightforward result, solving the inner problem Eq. (2.1b) directly with a FB algorithm is impractical.

By definition, the proximal operator is defined as follows:

$$\operatorname{prox}_{\eta\|\boldsymbol{\Omega}^\top \cdot\|_1}(\boldsymbol{w}) \triangleq \underset{\boldsymbol{x}\in\mathbb{R}^p}{\arg\min}\, \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{w}\|_2^2 + \eta\|\boldsymbol{\Omega}^\top \boldsymbol{x}\|_1 \ , \tag{B.3}$$

where $\eta > 0$ is usually the step size of the used FB algorithm. One notices that for $\eta = 1$, this problem is the same inner problem Eq. (2.1b). For other values of η, one can still show that both problems are equivalent by noticing that $\eta\|\boldsymbol{\Omega}^\top \boldsymbol{x}\|_1 = \|(\eta\boldsymbol{\Omega})^\top \boldsymbol{x}\|_1$. The difficulty in evaluating the proximal operator stems from the fact that the solution of Eq. (B.3) doesn't have a closed-form expression, as explained in the main script.

In practice, iterative optimizers are implemented to evaluate $\operatorname{prox}_{\eta\|\boldsymbol{\Omega}^\top \cdot\|_1}$, *e.g.,* writing the dual problem of Eq. (B.3) and solving it with any FB algorithm Chambolle et al. (2010). In our work, we choose to apply such optimizer directly on the inner problem based on the equivalence between Eqs. (2.1b) and (B.3). This saves us from introducing Eq. (B.3) as another nested optimization that would prohibitively increase the computation cost of our method, while yielding no benefits to the quality of denoised signals.

Annexe C

Résumé des travaux

C.1 Contexte

Le coût élevé de l'étiquetage des données représente un défi car la quantité de données générées augmente de façon exponentielle. En conséquence, il est courant d'observer à la fois des points de données étiquetés et non étiquetés, ces derniers étant généralement la grande majorité. Les tâches d'apprentissage sur des ensembles de données qui comprennent des points étiquetés et non étiquetés sont appelées apprentissage semi-supervisé (SSL). Le SSL est généralement traité avec des hypothèses supplémentaires sur les données. La principale, appelée *homophilie*, se réfère au fait que les points "proches" ont probablement des étiquettes similaires (Wang and Zhang, 2006). De plus, les points dans de nombreuses applications représentent des entités qui sont naturellement liées les unes aux autres, *e.g.,* en biologie (Liu et al., 2018) ou dans les médias sociaux (Liben-Nowell and Kleinberg, 2003). Là encore, les entités liées ont probablement la même étiquette, ce qui souligne l'importance d'exploiter les liens lors de la résolution de problèmes SSL. Par conséquent, diverses méthodes basées sur les graphes ont été développées pour le SSL.

Un problème avec de telles méthodes est que leurs performances dépendent fortement de la qualité du graphe. Ce problème pose un défi important car les graphes du monde réel sont intrinsèquement bruités, ce qui dégrade considérablement les performances. Dans ce travail, nous nous concentrons sur le problème de l'apprentissage de graphes de haute qualité pour des tâches d'apprentissage semi-supervisé. Dans la prochaine section, nous formulons le problème d'apprentissage semi-supervisé basé sur le graphe et discutons des approches existantes pour le résoudre. Puis, nous formulons le problème de l'apprentissage de graphes que nous

abordons. Ensuite, nous passons en revue les travaux existants sur l'apprentissage de graphes, et enfin nous présentons nos contributions.

C.2 Apprentissage semi-supervisé basé sur le graphe

Un graphe $\mathcal{G}$ est une paire (V, E), où V est un ensemble de n nœuds et $E \subseteq V \times V$ est un ensemble d'arêtes. Nous représentons un graphe par sa matrice d'adjacence $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, où $\boldsymbol{A}_{i,j}$ est le poids de l'arête entre les nœuds i et j. Nous désignons par $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ la matrice de caractéristiques dont les lignes incluent les caractéristiques des nœuds correspondants, et par $\boldsymbol{Y} \in \mathbb{R}^n$ le vecteur des étiquettes des nœuds.

Nous nous intéressons aux problèmes d'apprentissage semi-supervisé transductifs, où nous avons un ensemble de points, dont un sous-ensemble est étiqueté, et l'objectif est d'approximer la fonction d'étiquetage sur les points non étiquetés. Formellement, nous avons $(\boldsymbol{X},\ \mathcal{G}_{obs},\ \boldsymbol{Y}_{obs})$, où $\boldsymbol{X}$ représente les caractéristiques des nœuds, $\mathcal{G}_{obs}$ est le graphe observé, et $\boldsymbol{Y}_{obs} \in \mathbb{R}^n$ contient les étiquettes d'un sous-ensemble de points aux coordonnées $i \in V_{tr} \subset V$ et, par exemple, la valeur "NaN" en dehors de V_{tr}. Il existe essentiellement deux stratégies principales pour résoudre les problèmes d'apprentissage semi-supervisé basés sur les graphes. La première consiste à *propager* les étiquettes connues en utilisant un processus de *régularisation*. Les étiquettes prédites sont données par la formule suivante :

$$\boldsymbol{Y}_{\text{Reg}} \in \underset{\boldsymbol{Y} \in \mathcal{B}}{\arg\min} \frac{1}{|V_{tr}|} \sum_{i \in V_{tr}} \ell(\boldsymbol{Y}_i, (\boldsymbol{Y}_{obs})_i) + \lambda R(\boldsymbol{Y}, \boldsymbol{A}) \ , \tag{C.1}$$

où $\mathcal{B}$ est un ensemble admissible, ℓ est généralement une fonction de coût régulière, R est un terme de régularisation et λ est un paramètre de régularisation. Dans les tâches de régression, $\mathcal{B}$ est généralement l'espace $\mathbb{R}^n$ et ℓ est choisie comme étant l'erreur quadratique moyenne (MSE). En revanche, dans les tâches de classification, ℓ est l'entropie croisée catégorielle (CCE) et le i-ème élément $\boldsymbol{Y}_i$ n'est pas un scalaire mais plutôt un vecteur contenant la distribution de probabilité sur les classes. Formellement, $\mathcal{B} = \{\boldsymbol{Y} \in \mathbb{R}^{n \times C} \mid \forall i,\ \sum_{c=1}^{C} \boldsymbol{Y}_{i,c} = 1,\ \forall i, c,\ \boldsymbol{Y}_{i,c} \geq 0\}$, où C est le nombre de classes. Un choix populaire que nous considérons dans ce travail est la *régularisation laplacienne* (Slepcev and Thorpe, 2019) :

$$R(\boldsymbol{Y}, \boldsymbol{A}) = \frac{1}{|E|} \sum_{i,j} \boldsymbol{A}_{i,j} \|\boldsymbol{Y}_i - \boldsymbol{Y}_j\|_2^2 = \begin{cases} \frac{1}{|E|} \boldsymbol{Y}^\top \boldsymbol{L} \boldsymbol{Y} & \text{régression,} \\ \frac{1}{|E|} \sum_{c=1}^{C} (\boldsymbol{Y}_{:,c})^\top \boldsymbol{L} \boldsymbol{Y}_{:,c} & \text{classification,} \end{cases} \tag{C.2}$$

où $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ est le Laplacien du graphe, $\boldsymbol{D}$ est la matrice diagonale des degrés : $\boldsymbol{D}_{i,i} = \sum_j \boldsymbol{A}_{i,j}$ et $\boldsymbol{Y}_{:,c}$ est la c-ème colonne de $\boldsymbol{Y}$.

La deuxième stratégie principale pour l'apprentissage semi-supervisé est d'entraîner un *modèle paramétrique* $\boldsymbol{Y}_W(\boldsymbol{X}, \boldsymbol{A})$ paramétré par les poids W, tels que les GNNs. L'objectif s'écrit comme suit :

$$\begin{aligned} &\boldsymbol{Y}_{GNN}(\boldsymbol{A}) = \boldsymbol{Y}_{W^\star}(\boldsymbol{X}, \boldsymbol{A}), \text{ où} \\ &W^\star = \arg\min_W \frac{1}{|V_{tr}|} \sum_{i \in V_{tr}} \ell\Big(\big(\boldsymbol{Y}_W(\boldsymbol{X}, \boldsymbol{A})\big)_i, (\boldsymbol{Y}_{obs})_i \Big) \ . \end{aligned} \tag{C.3}$$

Dans ce travail, nous utilisons des GNN à passage de messages avec agrégation de la somme. La première couche est $\boldsymbol{X}^{[0]} = \boldsymbol{X}$, propagée comme suit :

$$\boldsymbol{X}^{[l]} = \phi(\boldsymbol{X}^{[l-1]}\boldsymbol{W}_1^{[l]} + \boldsymbol{A}\boldsymbol{X}^{[l-1]}\boldsymbol{W}_2^{[l]} + \boldsymbol{1}_n(\boldsymbol{b}^{[l]})^\top) \ , \tag{C.4}$$

où $\boldsymbol{W}_1^{[l]}, \boldsymbol{W}_2^{[l]} \in \mathbb{R}^{d_{l-1} \times d_l}, \boldsymbol{b}^{[l]} \in \mathbb{R}^{d_l}$ sont des poids ajustables, d_l est la dimensionnalité de sortie de la l-ème couche, $\boldsymbol{1}_n = (1, \ldots, 1)^\top \in \mathbb{R}^n$, et ϕ est une fonction non linéaire appliquée élément par élément. La sortie $\boldsymbol{Y}_W(\boldsymbol{X}, \boldsymbol{A}) = \boldsymbol{X}^{[k]}$ est obtenue après k itérations de propagation de messages, et les paramètres sont rassemblés dans $W = \{\boldsymbol{W}_1^{[l]}, \boldsymbol{W}_2^{[l]}, \boldsymbol{b}^{[l]}\}_{l=1}^k$.

C.2.1 Optimisation bi-niveau pour l'apprentissage de graphes

Dans cette thèse, nous nous concentrons sur le problème de l'apprentissage de graphes pour les méthodes SSL basées sur les graphes. Nous considérons le cas où la fonction objectif du graphe est une fonction du modèle basé sur le graphe *entraîné*, c'est-à-dire que nous examinons une *optimisation bi-niveau*. En utilisant un deuxième ensemble de nœuds étiquetés $V_{out} \subset V$ distinct de V_{tr} et étant donné un ensemble de matrices d'adjacence admissibles $\mathcal{A}$, l'optimisation bi-niveau est formulée comme suit :

$$\boldsymbol{A}^\star \in \arg\min_{\boldsymbol{A} \in \mathcal{A}} F_{out} = \tfrac{1}{|V_{out}|} \sum_{i \in V_{out}} \ell(\boldsymbol{Y}(\boldsymbol{A})_i, (\boldsymbol{Y}_{obs})_i), \tag{C.5}$$

tel que $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{GNN}(\boldsymbol{A})$ ou $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A})$. Autrement dit, le problème d'apprentissage de graphe, appelé problème *externe*, est une optimisation contrainte, où sa contrainte implique la sortie du problème d'optimisation *interne* $\boldsymbol{Y}(\boldsymbol{A})$: soit (C.2) sur $\boldsymbol{Y}$ ou (C.3) sur W. Plusieurs modèles sont possibles pour $\mathcal{A}$:

- **Apprentissage complet** : $\mathcal{A} = [a, b]^{n \times n}$ est l'ensemble de toutes les matrices d'adjacence pondérées (généralement avec des bornes a, b sur les poids). Ce choix conduit nécessairement à une complexité quadratique impraticable pour la minimisation.

- **Affinement des arêtes** : nous n'apprenons les poids que sur les arêtes existantes :

$$\mathcal{A} = \{\boldsymbol{A} \in [a,b]^{n\times n} | \boldsymbol{A}_{i,j} = 0 \text{ lorsque } (\boldsymbol{A}_{obs})_{i,j} = 0\}.$$

  La complexité est proportionnelle au nombre d'arêtes, généralement inférieure à quadratique en n car les graphes tendent à être épars.

- **Affinement généralisé des arêtes** : prise en compte d'un motif zéro obtenu par une modification de la matrice d'adjacence observée. Par exemple, le motif zéro de $\boldsymbol{A}^r_{obs}$ produit une arête entre les voisins qui sont à moins de r-hop l'un de l'autre dans $\mathcal{G}_{obs}$, où les nœuds i et j sont à r-hop l'un de l'autre si la longueur du chemin le plus court entre eux dans $\mathcal{G}_{obs}$ est de r.

Un autre modèle pour $\mathcal{A}$ peut être obtenu en adoptant **l'apprentissage métrique**, où le graphe appris est la sortie d'un modèle f_Θ paramétré par les poids Θ : $\mathcal{A} = \{\boldsymbol{A}_\Theta = f_\Theta(\boldsymbol{A}_{obs}, \boldsymbol{X})\}$. Dans ce scénario, le problème d'optimisation externe est effectué sur Θ au lieu de $\boldsymbol{A}$, c'est-à-dire que l'optimisation bi-niveau est formulée comme suit :

$$\Theta^\star \in \arg\min_\Theta F_{out} = \frac{1}{|V_{out}|} \sum_{i\in V_{out}} \ell(\boldsymbol{Y}(\boldsymbol{A}_\Theta)_i, (\boldsymbol{Y}_{obs})_i), \tag{C.6}$$

tel que $\boldsymbol{Y}(\boldsymbol{A}_\Theta) = \boldsymbol{Y}_{GNN}(\boldsymbol{A}_\Theta)$ ou $\boldsymbol{Y}(\boldsymbol{A}_\Theta) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A}_\Theta)$. À notre connaissance, le modèle d'apprentissage métrique n'a pas été considéré dans la littérature avec l'optimisation bi-niveau. En fait, l'adoption de ce modèle est l'une de nos contributions que nous mettons en évidence dans Appendix C.5.2.

C.3 Travaux connexes

Stretcu et al. (2019) ont proposé le modèle d'accord de graphe (*Graph Agreement Model*, GAM) pour l'apprentissage de graphes, qui a obtenu des résultats de pointe basé sur les graphes. GAM est un réseau profond entraîné à prédire la similarité entre les nœuds, c'est-à-dire l'apprentissage de métrique, en pénalisant l'absence d'une arête entre des nœuds ayant la même étiquette. Contrairement à GAM, Wang and Leskovec (2020) apprennent directement les poids des arêtes observées. L'optimisation implique l'apprentissage des paramètres du modèle GNN adopté. Étant donné que cela conduit à un surajustement en raison du grand nombre de paramètres, les auteurs utilisent le modèle de propagation d'étiquettes (*Label Propagation*, LP) (Zhu, 2005) pour régulariser le graphe. Le problème d'optimisation est formulé comme suit :

$$\min_{\boldsymbol{A},W} \frac{1}{|V_{tr}|} \sum_{i\in V_{tr}} \ell\Big(\big(\boldsymbol{Y}_W(\boldsymbol{X},\boldsymbol{A})\big)_i, (\boldsymbol{Y}_{obs})_i\Big) + R_{LP}(\boldsymbol{Y}_{obs}, \boldsymbol{A}) \ ,$$

où $\boldsymbol{Y}_W$ est la sortie du modèle GNN, $R_{LP}(\boldsymbol{Y}_{obs}, \boldsymbol{A})$ est le terme de régularisation basé sur LP. Le cadre proposé produit des résultats de pointe. Notez ici qu'une seule fonction objectif est utilisée pour évaluer conjointement le graphe et le modèle GNN. Nous désignons un tel paramétrage d'optimisation sous le nom d'*optimisation conjointe*. En général, l'optimisation conjointe est abordée à l'aide de méthodes basées sur les gradients. Nous soulignons que lors du calcul des gradients, $\boldsymbol{A}$ et W sont considérés comme des variables **indépendantes**, c'est-à-dire que $\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A}) = \boldsymbol{J}_{\boldsymbol{A}}(\boldsymbol{W}) = \boldsymbol{0}$, où $\boldsymbol{W}$ est une version vectorisée de W, et $\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A})$ est la matrice jacobienne $\big(\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A})\big)_{i,j} = \frac{\partial(\boldsymbol{W})_i}{\partial \boldsymbol{A}_j}$. De même, Fatemi et al. (2021) régularisent le graphe à l'aide d'un terme différent qui impose l'hypothèse selon laquelle un bon graphe doit également bien fonctionner pour le débruitage des caractéristiques des nœuds. De même, les mécanismes d'attention sont basés sur une optimisation conjointe, où après chaque couche GNN, les poids des arêtes sont réévalués en fonction de la similarité entre les représentations des nœuds dans cette couche. Le critère de similarité peut être défini par l'utilisateur, comme le produit scalaire (Luong et al., 2015; Vaswani et al., 2017), appris localement à chaque couche par un réseau feed-forward (Veličković et al., 2018), ou une combinaison des deux schémas (Kim and Oh, 2021). Ces mécanismes se sont révélés efficaces pour l'affinement des arêtes dans les tâches SSL.

Peu de travaux se sont attaqués à l'apprentissage de graphes en utilisant l'optimisation bi-niveau. Franceschi et al. (2019) apprennent les paramètres des distributions de probabilité de Bernoulli sur des arêtes aléatoires indépendantes. Ces paramètres sont optimisés pour minimiser la perte de validation sur un sous-ensemble de nœuds différent de celui utilisé pour entraîner le modèle GNN. Cette méthode inclut l'apprentissage de n^2 paramètres, ce qui limite sa scalabilité. De même, Wan and Kokel (2021) établissent une sparsification du graphe observé tout en maintenant sa connectivité en minimisant la perte de validation du modèle GNN. Cependant, cette méthode ne surpasse pas nécessairement le graphe observé. Les deux méthodes précédentes souffrent de ne pas généraliser aux nouveaux points, car cela nécessite de relancer le processus d'optimisation.

Une différence importante entre les paramétrages d'optimisation conjointe et bi-niveau lors de l'apprentissage du graphe à l'aide de méthodes basées sur les gradients est que $\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A}) \neq \boldsymbol{0}$ dans le cas du paramétrage bi-niveau, il est donc nécessaire d'incorporer $\boldsymbol{J}_{\boldsymbol{W}}(\boldsymbol{A})$ dans le calcul du gradient. Il s'agit du principal défi du paramétrage bi-niveau.

C.4 Différentiation itérative (ITD)

Les deux problèmes bi-niveaux dans les équations (C.6) et (C.5) sont intractables numériquement car ni la solution du problème interne ni son gradient par rapport à $\boldsymbol{A}$ ou à Θ n'ont une expression analytique qui puisse être évaluée. Ainsi, ni $\Theta^\star$ ni $\boldsymbol{A}^\star$ ne peuvent être évalués ou calculés itérativement par un algorithme du premier ordre. De plus, comme c'est généralement le cas dans l'apprentissage automatique moderne, le problème externe n'est pas convexe, nous ne cherchons donc pas à trouver son minimiseur, mais plutôt un bon ensemble de poids qui prouve l'efficacité de notre algorithme par rapport au graphe observé.

Récemment, des avancées ont été réalisées dans les algorithmes basés sur les gradients qui peuvent être appliqués dans de telles circonstances. La méthode que nous adoptons, appelée diffèrentiation itérative (*Iterative Differentiation*, ITD), est une approche importante dans ce paradigme (Domke, 2012; Maclaurin et al., 2015; Franceschi et al., 2017). Dorénavant, nous désignons le gradient externe ∇F_{out} sous le nom d'*hypergradient* afin de le distinguer du gradient interne ∇F_{in}.

À la recherche de simplicité, nous illustrons le fonctionnement de l'ITD sur Eq. (C.5). Étant donné un algorithme itératif qui converge vers la solution interne $\boldsymbol{Y}(\boldsymbol{A})$ ou vers une bonne approximation de celle-ci, et le nombre d'itérations internes τ_{in}, l'ITD consiste à remplacer $\boldsymbol{Y}(\boldsymbol{A})$ par la sortie de l'algorithme après τ_{in} itérations, puis à approximer l'hypergradient ∇F_{out} à l'aide de la *différentiation automatique* (*Automatic Differentiation*, AD) (Baydin et al., 2018; Verma, 2000). L'AD est une technique capable d'évaluer le gradient de la sortie d'un algorithme itératif par rapport aux variables d'entrée.

C.5 Contribution

La contribution de cette thèse est triple. Dans la première partie, nous abordons le problème de l'apprentissage d'aprioris parcimonieux de type analyse avec une optimisation bi-niveau en tant que point de départ simple. La deuxième partie présente un nouveau cadre pour l'apprentissage des structures de graphe, qui consiste à adopter l'apprentissage métrique dans le cadre bi-niveau. Cela signifie que nous entraînons un modèle paramétrique à prédire les poids des arêtes. Dans la dernière partie, nous identifions un manque de supervision induit lors de l'optimisation pour le graphe dans le cadre bi-niveau en mode de raffinement des arêtes. Nous donnons une caractérisation mathématique de ce phénomène pour différents modèles basés sur le graphe, et examinons des solutions possibles, y compris le cadre proposé dans la deuxième partie.

C.5.1 Apprentissage d'aprioris parcimonieux de type analyse avec une optimisation bi-niveau

Le débruitage est un problème largement abordé qui apparaît dans de nombreux domaines, allant de l'ingénierie biomédicale (McCann et al., 2019) à la vision par ordinateur (Yang et al., 2017) en passant par la télédétection (Addesso et al., 2017). L'objectif est de restaurer le signal à partir d'une observation bruitée. Généralement, le modèle du système d'imagerie est *a priori* connu : $\boldsymbol{y} = \boldsymbol{w} + \boldsymbol{\varepsilon}$, où $\boldsymbol{w}, \boldsymbol{y} \in \mathbb{R}^p$ sont respectivement les signaux réels et mesurés, et $\boldsymbol{\varepsilon} \in \mathbb{R}^p$ est un bruit additif. De plus, une hypothèse préalable sur la nature du signal peut être disponible, comme la parcimonie (McCann and Ravishankar, 2020). Cette connaissance supplémentaire peut être incorporée dans le processus d'optimisation pour obtenir de meilleures reconstructions, par exemple un rapport signal sur bruit plus élevé.

Les aprioris de parcimonie existent sous deux formes (Elad et al., 2007) : *i)* la parcimonie de type synthèse (traditionnelle) où $\boldsymbol{w} = \boldsymbol{\Omega}\boldsymbol{u}$, $\boldsymbol{\Omega}$ est un opérateur linéaire, et $\boldsymbol{u}$ est parcimonieux ; *ii)* la parcimonie de type analyse où $\boldsymbol{v} = \boldsymbol{\Omega}^\top \boldsymbol{w} \in \mathbb{R}^m$ est parcimonieux. Cette section s'intéresse à ce dernier cas. En tant que substitut convexe, cet apriori est imposée sur $\boldsymbol{w}$ en ajoutant le terme $\|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1$ à la fonction de perte (généralement quadratique) (Mancera and Portilla, 2006), où $\|\cdot\|_1$ est la norme ℓ_1. En rassemblant tous les éléments, le problème consiste à trouver : $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1$ pour une valeur de paramètre d'amplitude de régularisation $\lambda \geq 0$. L'opérateur linéaire $\boldsymbol{\Omega}$ est soit défini par l'utilisateur, soit appris directement à partir des données.

Le principal problème que nous abordons dans cette partie est d'extraire à la fois $\boldsymbol{\Omega}$ et λ à partir des données pour des tâches de débruitage avec apprentissage supervisé. Par conséquent, la tâche interne ici n'est pas un problème SSL (Semi-Supervised Learning) basé sur un graphe. En utilisant $\lambda \|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1 = \|(\lambda \boldsymbol{\Omega})^\top \boldsymbol{w}\|_1$, ce problème équivaut à extraire le produit $\lambda\boldsymbol{\Omega}$ en tant qu'objet unique, nous arrêtons donc d'écrire λ explicitement à partir de maintenant et nous conservons $\boldsymbol{\Omega}$. Formellement, la tâche qui nous intéresse est la suivante : étant donné un ensemble de données $(\boldsymbol{y}_l, \boldsymbol{w}_l)_{l=1}^L$ de L paires de mesures et de signaux réels associés, trouver l'opérateur $\boldsymbol{\Omega}$ qui minimise l'erreur quadratique moyenne entre les reconstructions et les signaux réels :

$$\min_{\boldsymbol{\Omega}\in\mathcal{A}} \sum_{l=1}^{L} \left\| \hat{\boldsymbol{w}}\left(\boldsymbol{\Omega}, \boldsymbol{y}_l\right) - \boldsymbol{w}_l \right\|_2^2 \tag{C.7a}$$

$$\text{où} \quad \hat{\boldsymbol{w}}(\boldsymbol{\Omega}, \boldsymbol{y}) = \arg\min_{\boldsymbol{w}\in\mathbb{R}^p} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{w}\|_2^2 + \|\boldsymbol{\Omega}^\top \boldsymbol{w}\|_1 \ . \tag{C.7b}$$

Eq. (C.7) est un *problème d'optimisation bi-niveau* : dans le problème externe, nous optimisons le dictionnaire $\boldsymbol{\Omega}$, tandis que dans le problème interne, nous débruitons les mesures. Nous savons déjà que la partie interne peut être résolue en appliquant l'algorithme de décomposition avant-arrière (*Forward-Backward*, FB) sur le problème dual (Chambolle et al., 2010). Cependant, en raison de la norme ℓ_1, ni la solution ni son gradient par rapport à $\boldsymbol{\Omega}$ n'ont une expression analytique. Ainsi, la solution du problème bi-niveau ne peut pas être dérivée analytiquement ni obtenue avec des méthodes basées sur les gradients.

Contribution : nous estimons l'opérateur analyse en utilisant ITD en développant l'algorithme FB appliqué sur le problème dual de Eq. (C.7b). C'est, à notre connaissance, le premier travail qui utilise l'ITD pour apprendre des aprioris parcimonieux de type analyse comme dans Eq. (C.7), qui est fortement non-lisse, sans recourir à une technique de relaxation. Cela permet d'examiner la capacité de l'ITD, en particulier de la phase d'AD, dans un tel contexte. En effet, les expériences prouvent l'efficacité de l'ITD dans l'apprentissage de l'opérateur à partir de signaux unidimensionnels constants par morceaux et d'images bidimensionnelles. De plus, nous proposons de réduire l'ensemble admissible $\mathcal{A}$ aux dictionnaires dont les colonnes somment à zéro lors de l'apprentissage à partir de signaux constants par morceaux, et nous prouvons empiriquement que cela augmente la stabilité et extrait un opérateur de meilleure qualité qu'une méthode de référence précédente.

Pourquoi ce problème comme point de départ ?

Comme indiqué précédemment, ce problème sert de point de départ avant de passer au problème de l'apprentissage de graphes. Cela est dû aux raisons suivantes :

- Les mises à jour FB au niveau interne sont disponibles sous une forme analytique, ce qui nécessite l'utilisation de la différentiation automatique (AD) pour effectuer une différentiation du premier ordre afin de calculer l'hypergradient. En revanche, pour le cas de l'apprentissage de graphes, le problème interne est optimisé à l'aide d'une méthode basée sur les gradients avec des gradients évalués via AD. Autrement dit, AD pour l'apprentissage de graphes nécessite l'évaluation des *gradients des gradients*. Par conséquent, (C.7) est un bon point de départ pour se familiariser avec l'utilisation de l'AD dans les problèmes bi-niveaux.

- Le problème (C.7) est non lisse, ce qui constitue un cadre difficile pour l'ITD. Étant donné que Peyré and Fadili (2011) ont relaxé Eq. (C.7) et ont dérivé une formule de l'hypergradient, nous pouvons valider les performances de l'ITD dans ce cadre en comparant sa sortie à l'hypergradient du problème relaxé.
- Le problème (C.7) est effectivement utilisé dans le contexte de l'apprentissage de graphes dans de nombreuses applications. En fait, lorsque les coefficients des signaux $\boldsymbol{w}_l$ se trouvent sur un graphe et sont connus pour être constants dans les voisinages, ce qui signifie que $\{(\boldsymbol{w}_l)_i - (\boldsymbol{w}_l)_j\}_{(i,j)\in E}$ est parcimonieux, (C.7) est utilisé pour apprendre le graphe en apprenant sa matrice d'incidence, c'est-à-dire que $\mathcal{A}$ est l'ensemble de toutes les matrices d'incidence des m arêtes.

Travaux connexes : Le problème (C.7) a été formulé pour la première fois dans Peyré and Fadili (2011), où les auteurs ont régularisé la norme ℓ_1 de manière à ce que l'hypergradient ait une expression analytique. De même, dans Sprechmann et al. (2013), un régime de régularisation différent est adopté. Cependant, la parcimonie recherchée est dégradée avec des régimes similaires qui régularisent ℓ_1 à *zéro* (Nikolova, 2000). Récemment, dans McCann and Ravishankar (2020), une formule de l'hypergradient a été dérivée sous certaines conditions, mais elle nécessite l'inversion itérative d'une grande matrice pour chaque point de données, ce qui rend son implémentation peu pratique. Chambolle and Pock (2021) réalisent une analyse de sensibilité pour calculer les hypergradients afin d'apprendre des dictionnaires de type convolution avec un petit support pour des signaux constants par morceaux. De telles contraintes fortes ne sont pas prises en compte dans notre travail ; cependant, nous montrons empiriquement qu'un simple recentrage des colonnes suffit pour apprendre un dictionnaire de haute qualité. Enfin, la différence majeure par rapport aux méthodes mentionnées précédemment réside dans l'utilisation de l'AD pour obtenir des hypergradients dans un tel *cadre non lisse*, sans utiliser de techniques de régularisation, ni dériver analytiquement un algorithme produisant ce gradient lorsqu'il est défini. Nous montrons avec des résultats empiriques la capacité de cette approche.

C.5.2 Apprentissage de modèles de graphe à graphe avec l'optimisation bi-niveau

Dans la deuxième partie de cette thèse, nous abordons l'apprentissage de graphes pour les méthodes d'apprentissage semi-supervisées. Plus précisément, nous nous intéressons au cas où l'objectif guidant le processus d'apprentissage est d'améliorer les performances du modèle basé sur le graphe entraîné dans la tâche SSL. C'est-

à-dire que nous nous intéressons à une *optimisation bi-niveau*.

L'idée directe consiste à utiliser l'optimisation bi-niveau (C.5) dans le cadre de l'affinement des arêtes ou de l'apprentissage complet. Pour le premier cas, nous montrons dans l'Appendix C.5.3 que cela induit un phénomène de manque de supervision que nous appelons *gradient scarcity*, qui dégrade notablement la qualité du graphe appris. En revanche, le dernier cas n'est pas pratique en raison de la complexité quadratique en le nombre de nœuds n.

À la place, nous proposons de résoudre le problème d'optimisation bi-niveau dans le cadre de l'apprentissage de la métrique, comme dans Eq. (C.6). En d'autres termes, nous entraînons un *modèle paramétrique* pour apprendre une métrique de similarité par paires entre les nœuds. Ce modèle prend en entrée les caractéristiques d'une paire de nœuds et le poids d'arête observé entre eux, et produit le poids d'arête optimisé en sortie. Nous appelons ce modèle G2G (Graph to Graph), en nous inspirant des modèles Set2Graph (Serviansky et al., 2020). En effet, lorsque le graphe observé est sans arête, G2G est une fonction qui transforme des ensembles en graphes.

Structure du modèle G2G : notre modèle proposé peut être exprimé comme suit

$$(\boldsymbol{A}_\Theta)_{i,j} = \gamma\Big(\beta\big(\alpha(\boldsymbol{X}_i), \alpha(\boldsymbol{X}_j)\big), (\boldsymbol{A}_{obs})_{i,j}\Big) \ , \tag{C.8}$$

où :

- l'encodeur $\alpha : \mathbb{R}^p \to \mathbb{R}^{p_\alpha}$ est un Multi-Layer Perceptron (MLP) qui calcule un nouveau vecteur de représentation de dimension p_α pour son nœud d'entrée. Un MLP est un réseau composé de plusieurs couches entièrement connectées. Ainsi, étant donné les caractéristiques de sortie $\boldsymbol{X}^{[l]}$ de la l-ème couche, la sortie de la couche suivante est calculée comme suit :

$$\boldsymbol{X}^{[l+1]} = \phi^{[l+1]}(\boldsymbol{X}^{[l]}\boldsymbol{W}_1^{[l+1]} + \mathbf{1}_n(\boldsymbol{b}^{[l+1]})^\top) \ , \tag{C.9}$$

  où $\boldsymbol{W}_1^{[l+1]} \in \mathbb{R}^{d_l \times d_{l+1}}, \boldsymbol{b}^{[l+1]} \in \mathbb{R}^{d_{l+1}}$ sont des paramètres ajustables, d_l est la dimension de sortie de la l-ème couche, et ϕ est la fonction d'activation non linéaire.

- l'agrégateur $\beta : \mathbb{R}^{p_\alpha} \times \mathbb{R}^{p_\alpha} \to \mathbb{R}^{p_\alpha}$ prend les plongements $\tilde{\boldsymbol{X}}_i, \tilde{\boldsymbol{X}}_j$ d'une paire de nœuds i, j et calcule

$$\beta : (\tilde{\boldsymbol{X}}_i, \tilde{\boldsymbol{X}}_j) \mapsto (\tilde{\boldsymbol{X}}_i - \tilde{\boldsymbol{X}}_j)^2, \tag{C.10}$$

  où $(\cdot)^2$ est la fonction carrée appliquée à chaque dimension. On remarque que cette fonction est invariante par rapport à l'ordre de ses entrées.

- le régresseur $\gamma : \mathbb{R}^{p_\alpha} \times \mathbb{R} \to [0, 1]$ est un MLP avec la fonction d'activation sigmoïde en sortie, qui produit le poids d'arête prédit.

L'architecture G2G proposée peut facilement être montrée comme étant *permutation-équivariante*, c'est-à-dire que la permutation du graphe d'entrée permute de la même manière le graphe de sortie. La permutation-équivariance (ou invariance) est une caractéristique essentielle à appliquer dans le traitement des graphes. Elle garantit que la structure du graphe de sortie est indépendante du reétiquetage des points dans l'ensemble de données et peut *généraliser à de nouveaux graphes*.

Contribution : nous proposons d'entraîner le modèle G2G en résolvant Eq. (C.6) via ITD. Remarquons ici que la différentiation automatique évalue les *gradients des gradients* car elle : *i)* calcule $\nabla_{\boldsymbol{W}} F_{in}$ pour les mises à jour internes, *ii)* évalue le gradient de la sortie après ces mises à jour par rapport à Θ. Nous optimisons ensuite Θ à l'aide d'un algorithme du premier ordre. À notre connaissance, il s'agit de la première étude qui entraîne un modèle G2G en utilisant l'optimisation bi-niveau. Contrairement à l'optimisation directe du graphe, le modèle G2G entraîné peut ensuite être utilisé pour reconstruire un graphe de haute qualité, même lors de l'ajout de nouveaux points à l'ensemble de données. Des expériences sur des ensembles de données SSL démontrent que notre cadre surpasse considérablement les modèles opérant sur le graphe observé.

Le lecteur constatera dans l'Appendix C.5.3 que, en plus des avantages précédemment mentionnés de l'utilisation de G2G par rapport à l'optimisation directe du graphe, les modèles G2G atténuent le problème de gradient scarcity qui se manifeste dans les tâches d'affinement des arêtes sur les bases de données de problèmes SSL.

C.5.3 Gradient scarcity dans l'apprentissage de graphes avec optimisation bi-niveau

Dans cette partie, nous étudions en détail le problème du *gradient scarcity* qui apparaît lors de la résolution de Eq. (C.5) dans le cadre de l'affinement des arêtes. le gradient scarcity se réfère au fait que les arêtes entre les nœuds non étiquetés "éloignés" des nœuds étiquetés reçoivent des *gradients nuls*, c'est-à-dire qu'elles ne reçoivent aucune supervision pendant l'optimisation.

Fatemi et al. (2021) ont observé le gradient scarcity lors de l'apprentissage du graphe et d'un modèle GNN par une optimisation conjointe. En effet, un GNN à k couches calcule l'étiquette d'un nœud en utilisant des informations provenant de nœuds situés à une distance au plus k de celui-ci. Cette étiquette ne dépend donc pas des arêtes reliant les nœuds en dehors de ce voisinage, et le terme de perte correspondant à cette étiquette renvoie des gradients nuls sur ces arêtes lointaines.

En d'autres termes, le gradient scarcity est dû au champ récepteur de profondeur finie des GNN à propagation de messages.

Cependant, il n'est pas évident de généraliser cet argument à l'optimisation bi-niveau (C.5) dans le cadre de l'affinement des arêtes. Plus précisément, la discussion précédente suppose que les poids entraînés du GNN après des mises à jour basées sur les gradients ne dépendent pas de la matrice d'adjacence $\boldsymbol{A}$, ce qui n'est pas le cas dans l'optimisation bi-niveau. De plus, si le problème se pose dans le cadre bi-niveau, les rôles de V_{tr} et V_{out} doivent être précisés. Une autre question concerne la résolution de ce problème en recourant à des modèles basés sur les graphes avec un champ récepteur infini, par exemple la régularisation laplacienne.

Contributions : nous prouvons que le gradient scarcity se produit dans le cadre de l'optimisation bi-niveau lors de l'utilisation de GNN en tant que classificateur. Nous montrons qu'en utilisant un GNN à k couches, les hypergradients sont nuls sur les arêtes entre les nœuds situés à au moins k-hop des nœuds étiquetés dans $V_{tr} \cup V_{out}$. Pour la régularisation laplacienne, nous prouvons que le problème persiste, car les hypergradients sont atténués de manière exponentielle avec la distance par rapport aux nœuds étiquetés. Nous validons empiriquement nos résultats. Ensuite, nous testons trois stratégies possibles pour résoudre ce problème : l'apprentissage métrique avec les modèles G2G, la régularisation des graphes et l'affinement généralisé des arêtes. De plus, nous distinguons empiriquement le gradient scarcity du surajustement, dans le sens où résoudre le premier ne résout pas nécessairement le second. À notre connaissance, il s'agit de la première étude qui aborde mathématiquement le problème du gradient scarcity pour l'optimisation bi-niveau des graphes, et examine ce phénomène pour les modèles avec un champ récepteur infini.

Les principaux théorèmes que nous prouvons sont les suivants :

Hypergradient scarcity avec les GNN : Nous considérons l'optimisation bi-niveau (C.5) en adoptant la méthode GNN $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{GNN}(\boldsymbol{A})$.

Théorème C.5.1. *Soit $\boldsymbol{Y}_W$ un GNN à k couches paramétré par l'ensemble de poids W. Supposons que le problème d'optimisation interne soit résolu avec un algorithme du premier ordre. Alors, pour toute paire de nœuds i, j situés à au moins k-hop des nœuds de $V_{out} \cup V_{tr}$, nous avons $\frac{\partial F_{out}}{\partial \boldsymbol{A}_{i,j}} = \mathbf{0}$.*

Notez qu'avec les GNN, l'optimisation interne n'est pas un problème convexe, ce qui nécessite d'optimiser pour obtenir un bon minimum local. Cependant, notre analyse démontre que le hypergradient scarcity se produit précisément dans ce scénario, et n'exige pas que l'algorithme du premier ordre converge vers l'minimi-

seur du problème interne.

Hypergradient scarcity avec la régularisation laplacienne : Nous montrons que bien que dans une moindre mesure, ce problème se pose également lorsque l'on adopte la régularisation laplacienne, c'est-à-dire $\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{Y}_{\text{Reg}}(\boldsymbol{A})$. Plus précisément, nous établissons que l'amplitude de l'hypergradient diminue de manière *exponentielle* lorsque la somme des deux distances par rapport à V_{tr} et V_{out} augmente. Notre étude est axée sur les tâches de régression, où ℓ est la fonction de perte MSE dans les équations (C.1) et (C.5). Soit $\boldsymbol{S}_{in} \in \mathbb{R}^{n\times n}$ la matrice diagonale dont les entrées sont égales à 1 pour les nœuds de V_{tr} et 0 sinon, la solution $\boldsymbol{Y}(\boldsymbol{A})$ bénéficie d'une expression analytique :

$$\boldsymbol{Y}(\boldsymbol{A}) = \boldsymbol{B}^{-1}\tilde{\boldsymbol{S}}_{in}\boldsymbol{Y}_{obs} \ ,$$

où $\tilde{\boldsymbol{S}}_{in} = \frac{\boldsymbol{S}_{in}}{|V_{tr}|}$ et $\boldsymbol{B} = \frac{\boldsymbol{S}_{in}}{|V_{tr}|} + \lambda\frac{\boldsymbol{L}}{|E|}$. Maintenant que nous avons cela, nous énonçons le résultat principal pour le scénario de régularisation laplacienne.

Théorème C.5.2. *Soient les nœuds i, j situés à au moins k-hop de V_{out}, et q-hop de V_{tr}. Alors nous avons :*

$$\left|\frac{\partial F_{out}}{\partial \boldsymbol{A}_{ij}}\right| \lesssim \lambda\frac{\sqrt{|V_{out}|} + \mu_{\min}\sqrt{|V_{tr}|}|V_{out}|}{\mu_{\min}^3|V_{tr}||E|}y_\infty^2(1-\mu)^{q+k} \ ,$$

où $\mu_{\min}$ ($\mu_{\max}$) est la plus petite (la plus grande) valeur propre de $\boldsymbol{B}$ que nous prouvons satisfaire $0 < \mu_{\min} < \mu_{\max}$, $\mu = \frac{\mu_{\min}}{\mu_{\max}}$, et $y_\infty = \|\boldsymbol{Y}_{obs}\|_\infty$.

Puisque $0 < 1 - \mu < 1$, le Théorème C.5.2 indique que l'hypergradient est exponentiellement atténué lorsque $q + k$ augmente.

Bibliography

P. Addesso, M. Dalla Mura, L. Condat, R. Restaino, G. Vivone, D. Picone, and J. Chanussot. Hyperspectral image inpainting based on collaborative total variation. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 4282–4286, 2017.

Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48, 2013.

Firoj Alam, Shafiq Joty, and Muhammad Imran. Graph based semi-supervised learning with convolution neural networks to classify crisis related tweets. In *Proceedings of the international AAAI conference on web and social media*, volume 12, 2018.

Mohammad Aliannejadi, Masoud Kiaeeha, Shahram Khadivi, and Saeed Shiry Ghidary. Graph-based semi-supervised conditional random fields for spoken language understanding using unaligned data. *arXiv preprint arXiv:1701.08533*, 2017.

Muthu Balaanand, N Karthikeyan, S Karthik, R Varatharajan, Gunasekaran Manogaran, and CB Sivaparthipan. An enhanced graph-based semi-supervised learning algorithm to detect fake users on twitter. *The Journal of Supercomputing*, 75:6085–6105, 2019.

Jonathan F Bard. Coordination of a multidivisional organization through two levels of management. *Omega*, 11(5):457–468, 1983.

Dolgorsuren Batjargal, Kifayat Ullah Khan, and Young-Koo Lee. Em-fgs: Graph sparsification via faster semi-metric edges pruning. *Applied Intelligence*, 49: 3731–3748, 2019.

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18, 2018.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.*, 2(1):183–202, 2009.

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(11), 2006.

Omar Ben-Ayed, David E Boyce, and Charles E Blair III. A general bilevel linear programming formulation of the network design problem. *Transportation Research Part B: Methodological*, 22(4):311–318, 1988.

Kristin P Bennett, Jing Hu, Xiaoyun Ji, Gautam Kunapuli, and Jong-Shi Pang. Model selection via bilevel optimization. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1922–1929. IEEE, 2006.

Peter Berger, Gabor Hannak, and Gerald Matz. Graph signal recovery via primal-dual algorithms for total variation minimization. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):842–855, 2017.

Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data*, 1(1), 2007.

Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, 1998.

Jerome Bracken and James T McGill. Mathematical programs with optimization problems in the constraints. *Operations research*, 21(1):37–44, 1973.

Wilfred Candler and Roger D Norton. *Multi-level programming*, volume 20. World Bank, 1977.

Antonin Chambolle and Thomas Pock. Learning consistent discretizations of the total variation. *SIAM Journal on Imaging Sciences*, 14(2):778–813, 2021.

Antonin Chambolle, Vicent Caselles, Daniel Cremers, Matteo Novaga, and Thomas Pock. An introduction to total variation for image analysis. *Theoretical foundations and numerical methods for sparse recovery*, 9(263-340):227, 2010.

Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23(89):1–64, 2022.

Tianyi Chen, Yuejiao Sun, and Wotao Yin. Tighter analysis of alternating stochastic gradient method for stochastic nested problems. *arXiv preprint arXiv:2106.13781*, 2021.

Yu Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in neural information processing systems*, 33:19314–19326, 2020.

Yanwen Chong, Yun Ding, Qing Yan, and Shaoming Pan. Graph-based semi-supervised learning: A review. *Neurocomputing*, 408:216–230, 2020.

PA Clark. Bilevel programming for steady-state chemical process design—ii. performance study for nondegenerate problems. *Computers & Chemical Engineering*, 14(1):99–109, 1990.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

George F Corliss. Applications of differentiation arithmetic. In *Reliability in Computing*, pages 127–148. Elsevier, 1988.

Mathieu Dagréou, Pierre Ablin, Samuel Vaiter, and Thomas Moreau. A framework for bilevel optimization that enables stochastic and global variance reduction algorithms. *arXiv preprint arXiv:2201.13409*, 2022.

Samuel I Daitch, Jonathan A Kelner, and Daniel A Spielman. Fitting a graph to vector data. In *Proceedings of the 26th annual international conference on machine learning*, pages 201–208, 2009.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326. PMLR, 2012.

Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586, 2011.

Abolfazl Doostparast Torshizi and Linda R Petzold. Graph-based semi-supervised learning with genomic data integration using condition-responsive genes applied to phenotype classification. *Journal of the American Medical Informatics Association*, 25(1):99–108, 2018.

Michael Elad, Peyman Milanfar, and Ron Rubinstein. Analysis versus synthesis in signal priors. *Inverse problems*, 23(3):947, 2007.

Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems*, 34:22667–22681, 2021.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Rémi Flamary, Alain Rakotomamonjy, and Gilles Gasso. Learning constrained task similarities in graphregularized multi-task learning. *Regularization, Optimization, Kernels, and Support Vector Machines*, 103, 2014.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pages 1165–1173. PMLR, 2017.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR, 2018.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pages 1972–1982. PMLR, 2019.

Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley J Osher, and Wotao Yin. Fixed point networks: Implicit depth models with jacobian-free backprop. *ArXiv e-print*, 2021.

Steven A Gabriel, Antonio J Conejo, J David Fuller, Benjamin F Hobbs, and Carlos Ruiz. *Complementarity modeling in energy markets*, volume 180. Springer Science & Business Media, 2012.

Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

Hashem Ghanem, Nicolas Keriven, Joseph Salmon, and Samuel Vaiter. Supervised graph learning with bilevel optimization. In *18th International Workshop on Mining and Learning with Graphs*, 2022.

Hashem Ghanem, Joseph Salmon, Nicolas Keriven, and Samuel Vaiter. Supervised learning of analysis-sparsity priors with automatic differentiation. *IEEE Signal Processing Letters*, 30:339–343, 2023a.

Hashem Ghanem, Samuel Vaiter, and Nicolas Keriven. Gradient scarcity with bilevel optimization for graph learning. *arXiv preprint arXiv:2303.13964*, 2023b.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

Davis Gilton, Gregory Ongie, and Rebecca Willett. Deep equilibrium architectures for inverse problems in imaging. *IEEE Transactions on Computational Imaging*, 7:1123–1133, 2021.

Johannes Grabmeier and Erich Kaltofen. *Computer Algebra Handbook: Foundations, Applications, Systems*. Springer Science & Business Media, 2003.

Riccardo Grazzi, Luca Franceschi, Massimiliano Pontil, and Saverio Salzo. On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, pages 3748–3758. PMLR, 2020.

Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19, 2006.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

Pierre Hansen, Brigitte Jaumard, and Gilles Savard. New branch-and-bound rules for linear bilevel programming. *SIAM Journal on scientific and Statistical Computing*, 13(5):1194–1217, 1992.

Howard Heaton, Samy Wu Fung, Aviv Gibali, and Wotao Yin. Feasibility-based fixed point networks. *Fixed Point Theory and Algorithms for Sciences and Engineering*, 2021(1):1–19, 2021.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex analysis and minimization algorithms I: Fundamentals*, volume 305. Springer science & business media, 2013.

Chenhui Hu, Lin Cheng, Jorge Sepulcre, Georges El Fakhri, Yue M Lu, and Quanzheng Li. A graph theoretical regression model for brain connectivity learning of alzheimer's disease. In *2013 IEEE 10th International Symposium on Biomedical Imaging*, pages 616–619. IEEE, 2013.

Tony Jebara, Jun Wang, and Shih-Fu Chang. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 441–448, 2009.

Max E Jerrell. Automatic differentiation and interval arithmetic for estimation of disequilibrium models. *Computational Economics*, 10(3):295–316, 1997.

Kaiyi Ji. *Bilevel optimization for machine learning: Algorithm design and convergence analysis*. The Ohio State University, 2021.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2021.

Vassilis Kalofolias. How to learn a graph from smooth signals. In *Artificial Intelligence and Statistics*, pages 920–929. PMLR, 2016.

Vassilis Kalofolias and Nathanaël Perraudin. Large scale graph learning from smooth signals. *arXiv preprint arXiv:1710.05654*, 2017.

Ashish Kapoor, Hyungil Ahn, Yuan Qi, and Rosalind Picard. Hyperparameter and kernel learning for graph based semi-supervised classification. *Advances in neural information processing systems*, 18, 2005.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *J. Mach. Learn. Res.*, 21(70):1–73, 2020.

Nicolas Keriven. Not too little, not too much: a theoretical analysis of graph (over) smoothing. *arXiv preprint arXiv:2205.12156*, 2022.

Dokyoon Kim, Hyunjung Shin, Young Soo Song, and Ju Han Kim. Synergistic effect of different levels of genomic data for cancer clinical outcome prediction. *Journal of biomedical informatics*, 45(6):1191–1198, 2012.

Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2021.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Erich Kobler, Alexander Effland, Karl Kunisch, and Thomas Pock. Total deep variation for linear inverse problems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7549–7558, 2020.

Bruno Lecouat, Jean Ponce, and Julien Mairal. Designing and learning trainable priors with non-cooperative games. *ArXiv*, 2020.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 556–559, 2003.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. *Advances in neural information processing systems*, 31, 2018.

Risheng Liu, Jiaxin Gao, Jin Zhang, Deyu Meng, and Zhouchen Lin. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):10045–10067, 2021a.

Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*, 2021b.

Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022a.

Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. Towards unsupervised deep graph structure learning. In *Proceedings of the ACM Web Conference 2022*, pages 1392–1403, 2022b.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020.

Qing Lu and Lise Getoor. Link-based classification. In *ICML 2003*, 2003.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015.

L. Mancera and J. Portilla. L0-norm-based sparse representation through alternate projections. In *2006 International Conference on Image Processing*, pages 2089–2092, 2006.

Patrice Marcotte. Network design problem with congestion effects: A case of bilevel programming. *Mathematical programming*, 34(2):142–162, 1986.

Michael T McCann and Saiprasad Ravishankar. Supervised learning of sparsity-promoting regularizers for denoising. *arXiv preprint arXiv:2006.05521*, 2020.

Michael T McCann, Michael Unser, et al. Biomedical image reconstruction: From the foundations to deep neural networks. *Foundations and Trends® in Signal Processing*, 13(3):283–359, 2019.

Qiaozhu Mei, Duo Zhang, and ChengXiang Zhai. A general optimization framework for smoothing language models on graph structures. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 611–618, 2008.

Peyman Milanfar. A tour of modern image filtering: New insights and methods, both practical and theoretical. *IEEE signal processing magazine*, 30(1):106–128, 2012.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240, 2014.

Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 27–38, 2017.

Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, page 1, 2012.

Mark EJ Newman, Duncan J Watts, and Steven H Strogatz. Random graph models of social networks. *Proceedings of the national academy of sciences*, 99 (suppl_1):2566–2572, 2002.

Mila Nikolova. Local strong homogeneity of a regularized estimator. *SIAM Journal on Applied Mathematics*, 61(2):633–658, 2000.

Jiahao Pang and Gene Cheung. Graph laplacian regularization for image denoising: Analysis in the continuous domain. *IEEE Transactions on Image Processing*, 26 (4):1770–1785, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International conference on machine learning*, pages 737–746. PMLR, 2016.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

Gabriel Peyré and Jalal M Fadili. Learning analysis sparsity priors. In *Sampta'11*, pages 4–pp, 2011.

Lishan Qiao, Limei Zhang, Songcan Chen, and Dinggang Shen. Data-driven graph construction and graph learning: A review. *Neurocomputing*, 312:336–351, 2018.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

Saiprasad Ravishankar and Yoram Bresler. Sparsifying transform learning with efficient optimal updates and convergence guarantees. *IEEE Transactions on Signal Processing*, 63(9):2389–2404, 2015.

R Tyrrell Rockafellar. *Conjugate duality and optimization*. SIAM, 1974.

Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. *Advances in Neural Information Processing Systems*, 33:12559–12571, 2020.

Jun-Hyung Ryu, Vivek Dua, and Efstratios N Pistikopoulos. A bilevel programming framework for enterprise-wide process networks under uncertainty. *Computers & Chemical Engineering*, 28(6-7):1121–1129, 2004.

Lawrence K Saul and Sam T Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of machine learning research*, 4 (Jun):119–155, 2003.

Hadar Serviansky, Nimrod Segol, Jonathan Shlomi, Kyle Cranmer, Eilam Gross, Haggai Maron, and Yaron Lipman. Set2graph: Learning graphs from sets. *Advances in Neural Information Processing Systems*, 33:22080–22091, 2020.

Chenggen Shi, Jie Lu, and Guangquan Zhang. An extended kuhn–tucker approach for linear bilevel programming. *Applied Mathematics and Computation*, 162(1): 51–63, 2005.

Dejan Slepcev and Matthew Thorpe. Analysis of p-laplacian regularization in semisupervised learning. *SIAM Journal on Mathematical Analysis*, 51(3):2085–2120, 2019.

Yunsheng Song, Jing Zhang, and Chao Zhang. A survey of large-scale graph-based semi-supervised classification algorithms. *International Journal of Cognitive Computing in Engineering*, 3:188–198, 2022a.

Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. Graph-based semi-supervised learning: A comprehensive review. *IEEE Transactions on Neural Networks and Learning Systems*, 2022b.

Pablo Sprechmann, Roee Litman, Tal Ben Yakar, Alexander M Bronstein, and Guillermo Sapiro. Supervised sparse analysis and synthesis operators. *Advances in Neural Information Processing Systems*, 26:908–916, 2013.

Heinrich von Stackelberg. *Theory of the market economy*. Oxford University Press, 1952.

Ulrich Stelzl, Uwe Worm, Maciej Lalowski, Christian Haenig, Felix H Brembeck, Heike Goehler, Martin Stroedicke, Martina Zenkner, Anke Schoenherr, Susanne Koeppen, et al. A human protein-protein interaction network: a resource for annotating the proteome. *Cell*, 122(6):957–968, 2005.

Gilbert W Stewart. *Matrix algorithms: volume 1: basic decompositions*. SIAM, 1998.

Otilia Stretcu, Krishnamurthy Viswanathan, Dana Movshovitz-Attias, Emmanouil Platanios, Sujith Ravi, and Andrew Tomkins. Graph agreement models for semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Amarnag Subramanya, Slav Petrov, and Fernando Pereira. Efficient graph-based semi-supervised learning of structured tagging models. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 167–176, 2010.

Dazhi Sun, Rahim F Benekohal, and S Travis Waller. Bi-level programming formulation and heuristic solution approach for dynamic traffic signal optimization. *Computer-Aided Civil and Infrastructure Engineering*, 21(5):321–333, 2006.

Martin Szummer and Tommi Jaakkola. Partially labeled classification with markov random walks. *Advances in neural information processing systems*, 14, 2001.

Partha Pratim Talukdar and Koby Crammer. New regularized algorithms for transductive learning. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part II 20*, pages 442–457. Springer, 2009.

William Thomas Tutte and William Thomas Tutte. *Graph theory*, volume 21. Cambridge university press, 2001.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

Arun Verma. An introduction to automatic differentiation. *Current Science*, pages 804–807, 2000.

Guihong Wan and Harsha Kokel. Graph sparsification via meta-learning. *DLG@ AAAI*, 2021.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234, 2016.

Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. In *Proceedings of the 23rd international conference on Machine learning*, pages 985–992, 2006.

Hongwei Wang and Jure Leskovec. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755*, 2020.

Lirong Wu, Haitao Lin, Cheng Tan, Zhangyang Gao, and Stan Z Li. Self-supervised learning on graphs: Contrastive, generative, or predictive. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 1907–1913. AAAI Press, 2019. ISBN 9780999241141.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Mehrdad Yaghoobi, Sangnam Nam, Rémi Gribonval, and Mike E Davies. Noise aware analysis operator learning for approximately cosparse signals. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5409–5412. IEEE, 2012.

Bo Yang, Hongkai Wen, Sen Wang, Ronald Clark, Andrew Markham, and Niki Trigoni. 3d object reconstruction from a single depth view with adversarial learning. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 679–688, 2017.

Han Yang, Kaili Ma, and James Cheng. Rethinking graph regularization for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4573–4581, 2021.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 40–48. JMLR.org, 2016.

David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196, 1995.

Wei Ye, Zexi Huang, Yunqi Hong, and Ambuj Singh. Graph neural diffusion networks for semi-supervised learning. *arXiv preprint arXiv:2201.09698*, 2022.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

Xiaojin Zhu. *Semi-supervised learning with graphs*. Carnegie Mellon University, 2005.

Xiaojin Zhu and Zoubin Ghahramani. *Learning from labeled and unlabeled data with label propagation*. Citeseer, 2002.

Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. Deep graph structure learning for robust representations: A survey. *arXiv preprint arXiv:2103.03036*, 14, 2021.