Supervised graph learning with bilevel optimization *

Hashem Ghanem[1][0000−0003−1690−1316], Nicolas Keriven[2][0000−0002−3846−8763], Joseph Salmon[3][0000−0002−3181−0634], and Samuel Vaiter[4][0000−0002−4077−708X]

[1] CNRS and IMB, Université de Bourgogne, Dijon, France
[2] CNRS, GIPSA-lab, Grenoble, France
[3] IMAG, Université de Montpellier, CNRS, Montpellier, France
[4] CNRS and Laboratoire J. A. Dieudonné, Université Côte d'Azur, Nice, France

Abstract. Graph-based learning attracted attention recently due to its efficiency analyzing data lying on graphs. Unfortunately, graphs in real-world are usually either not-given, noisy, or incomplete. In this work, we design a novel algorithm that addresses this issue by training a $G2G$ (Graph to Graph) model with a *bilevel optimization* framework to learn a better graph in a supervised manner. The trained model operates not only on training data, but generalizes to unseen data points. A bilevel problem comprises two optimization problems, referred to as outer and inner problem. The inner problem aims to solve the downstream task, *e.g.,* training a GCN (Graph Convolutional Network) model, whereas the outer one introduces a new objective function to evaluate the inner model performance, and the $G2G$ model is trained to minimize this function. To solve this optimization, we replace the solution of the inner problem with the output of any gradient-based algorithm proven to give a good surrogate. Then, we use automatic differentiation to compute the gradient of this output *w.r.t.* the $G2G$ weights, which we consequently learn with a gradient-based algorithm. Experiments on semi-supervised learning datasets show that the graph learned by the $G2G$ model outperforms the original graph by a significant margin.

Keywords: Graph learning · Bilevel optimisation.

1 Introduction

Graph-based learning has received increasing attention since data lying on graph structures is ubiquitous in many fields, ranging from social networks [20] to knowledge base[14], chemistry and medicine [29, 21] and traffic [39]. Also see [6, 15]. In these domains, it is important to deploy graph structures together with the given features to extract the sought for information. In social networks for example, users form the nodes of a graph and friendship connections between them form its edges. The thoughts and beliefs of an individual are likely to

* Supported by ANR Grava ANR-18-CE40-0005 and ANR GRandMa ANR-21-CE23-0006.

be shared by his 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.

To exploit graph structures along with the features of samples (also called nodes in this context), many methods can be used. The graph Laplacian regularization introduces a penalty term to the objective function to promote smoothness on the graph [32, 27]. This approach proved to have a good performance in many reconstruction problems, *e.g.,* image filtering [24]. Another method is using Graph Convolutional Networks ($GCNs$) that adopts the message passing model, where a new embedding of each node is computed based on its features and the features of its neighbor nodes [12, 38]. $GCNs$ may also be interpreted as performing spectral filtering on graphs [5, 16]. Comparing the two procedures, graph Laplacian regularization promotes similarity between connected nodes but, unlike $GCNs$, is not a supervised-based method, which is why it generally yields lower performance.

However, 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 they were mutually independent. This degrades the performance and leads to a sub-optimal solution. In this work, we alleviate this difficulty solving a *bilevel optimization problem* to train a model on reconstructing graphs of higher quality, in a way that improves performance in *any* supervised or semi-supervised learning problem. We refer to this model by $G2G$ (Graph to Graph), named with inspiration from *Set2Graph* models [31]. Indeed when the graph is edgeless, G2G is a function from sets to graphs. In this bilevel framework, the inner problem is a semi-supervised (or supervised) learning problem using the $G2G$ output graph, while the outer problem optimizes the $G2G$ weights s.t. the trained model in the inner problem performs well *w.r.t.* a well-designed objective function. To our knowledge, this is the first work that trains a $G2G$ model through a bilevel optimization framework, where such trained model constructs the underlying graph not only for data used in training, but also generalizes to new points.

Notations: A graph $\mathcal{G}$ is a pair (V, E), where V is a set of n nodes and $E \subseteq V \times V$ is a set of edges. We represent a graph by its adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, where $\boldsymbol{A}_{i,j}$ is the weight of the edge between nodes i, j. We denote by $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ the features matrix whose rows include the features of corresponding nodes, and by $\boldsymbol{Y} \in \mathbb{R}^{n_s}$ the available labels of n_s nodes.

Problem of interest: Although our framework can be deployed in both supervised and Semi-Supervised Learning (SSL) settings, we choose to specialize in the range of SSL problems, where we have a set of data points, a subset of which is labelled, and the goal is to approximate the labelling function on unlabelled points. Formally, we have $(\boldsymbol{X},\ \mathcal{G}_{obs},\ \boldsymbol{Y})$, where $\mathcal{G}_{obs}$ is the observed graph, and $\boldsymbol{Y} \in \mathbb{R}^{n_s}$ contains the labels of a subset of points $V_s \subset V$. The task is to train a graph-based model parameterized by the weights W to predict labels $\hat{\boldsymbol{Y}}_W(\boldsymbol{X}, \boldsymbol{A_{obs}})$. To train such model, we search for a good realization of W that

minimizes an objective function F_{in}:

$$\arg\min_{W} F_{in}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}_W(\boldsymbol{X}, \boldsymbol{A_{obs}}), \boldsymbol{A_{obs}}) \ , \tag{1}$$

where F_{in} might have $\boldsymbol{A_{obs}}$ as an explicit input, *e.g.,* objective functions coupled with the Laplacian regularization as in Eq. (5).

However, real-world graphs are noisy or non-given, which degrades the solution of Eq. (1). In this work, we alleviate this problem by training a $G2G$ model to learn a pair-wise similarity metric between nodes to construct high-quality graphs. This boosts performance and induces better generalization for both structure refinement/inference problems. This implies that a new objective function F_{out} is designed to optimize the $G2G$ model. We consider the case where this objective is a function of the output of Eq. (1), after replacing $\boldsymbol{A}_{obs}$ by $G2G$'s output. So, we look at a *bilevel optimization*:

$$\Theta^\star \in \arg\min_{\Theta} F_{out}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}_{W_\Theta}(\boldsymbol{X}, \boldsymbol{A}_\Theta), \boldsymbol{A}_\Theta) \ , \tag{2a}$$

such that

$$W_\Theta = \arg\min_{W} F_{in}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}_W(\boldsymbol{X}, \boldsymbol{A}_\Theta), \boldsymbol{A}_\Theta) \ , \tag{2b}$$

where Θ is the weights of the $G2G$ model, $\boldsymbol{A}_\Theta = G2G(\boldsymbol{X}, \boldsymbol{A}_{obs})$ is its output adjacency matrix. More details on the $G2G$ structure is available in Section 3.1. We refer to Eq. (2a) by the outer problem and to Eq. (2b) by the inner problem.

Seeking better generalization, we split labeled nodes in two sets V_{tr_1} and V_{tr_2}, each is used to optimize one objective function. The outer objective function writes:

$$F_{out} = \frac{1}{|V_{tr1}|} \sum_{i \in V_{tr1}} \ell\Big(\big(\hat{\boldsymbol{Y}}_{W_\Theta}(\boldsymbol{X}, \boldsymbol{A}_\Theta)\big)_i, \boldsymbol{Y}_i \Big) \ , \tag{3}$$

where $V_{tr1} \subset V_s$ is the outer training set, ℓ is a loss function commonly chosen to be the Categorical Cross Entropy (CCE) loss for classification, and the Mean Square Error (MSE) for regression. One may add a regularization term to F_{out} to impose some regularity or priors on the generated graph, but this isn't considered here. Regarding the inner loss function, there are two main roles a graph can play in graph-based methods. The first of which is when the graph explicitly appears as input to the adopted model (*e.g.,* message passing models and spectral filtering), unlike the other role where the graph is used to regularize the model but not as input of it. To show the capacity of our method under both settings, we fairly choose a representative method for each role, namely GCN models and the Laplacian regularization, respectively. For a GCN model $\hat{\boldsymbol{Y}}_W(\boldsymbol{X}, \boldsymbol{A})$ with weights W, whose variant is described in Appendix A, this leads to:

$$F_{in} = \frac{1}{|V_{tr2}|} \sum_{i \in V_{tr2}} \ell\Big(\big(\hat{\boldsymbol{Y}}_W(\boldsymbol{X}, \boldsymbol{A}_\Theta)\big)_i, \boldsymbol{Y}_i \Big) \ , \tag{4}$$

whereas in the case of using the Laplacian regularization, it reads:

$$F_{in} = \frac{1}{|V_{tr2}|} \sum_{i \in V_{tr2}} \ell(\hat{\boldsymbol{Y}}_i, \boldsymbol{Y}_i) + \frac{\lambda}{|E|} \sum_{(i,j) \in E} (\boldsymbol{A}_\Theta)_{i,j} (\hat{\boldsymbol{Y}}_i - \hat{\boldsymbol{Y}}_j)^2, \tag{5}$$

where $V_{tr2} \subset V_s$ is the inner training set, and λ is the regularization magnitude. Note that we directly optimize for $\hat{\boldsymbol{Y}}$ when using the Laplacian regularization, unlike the GCN case where we have the weights W. However, we keep writing $\hat{\boldsymbol{Y}}_W$ for both cases seeking unity of notation.

The bilevel problem in Eq. (2) 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 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 given graph.

In the present paper, we propose to train the $G2G$ model by replacing the inner problem in Eq. (2b) by a repeated application of any gradient-based algorithm, that is guaranteed to converge to a good proxy. Then, we use automatic differentiation [1, 37], more precisely higher-order automatic differentiation through the `Higher` package [13], to trace these dynamics and finally compute $\nabla_\Theta F_{out}$. Remark here that the `Higher` package evaluates *gradients of gradients* as it: 1) computes $\nabla_{W_\Theta} F_{in}$ for the inner updates, 2) evaluates the gradient of the output after these updates *w.r.t.* Θ. We optimize Θ afterwards using a gradient-based algorithm, and the resulted $G2G$ model can be 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 updates, here 10 iterations, in the inner problem to get a good approximation $\nabla_\Theta F_{out}$, which significantly reduce memory and time costs. Experiments on SSL datasets prove that our framework considerably outperforms models operating on the observed graph.

2 Related work

Bilevel optimization is used in many applications like multi-task and meta learning [2, 9, 10]. See [7] for a review of applications in different fields. Graph structure learning, on the other hand, gained in importance since the success of $GCNs$ in relational learning, stemming from the fact that real-world graphs usually have corrupted edges. The first way used to construct these graphs might be the k-nearest neighbors technique [30, 34] and its variants, but with shortcomings: we have to choose k and the associated similarity criterion. Here, we look at situations where the graph learning problem can be naturally formulated as a *supervised* bilevel optimization problem.

In [11], 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 GCN's validation loss. This method includes learning n^2 parameters which limits scalability, and it suffers from not generalizing to new points, as this requires re-running the optimization process to learn the parameters of added points. 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 doesn't depend on the dataset size, but rather on the problem complexity.

In [33], a method referred to as Graph Agreement Model (GAM) is trained to learn graphs for SSL problems by penalizing the absence of an edge between nodes with the same label, thus it isn't explicitly a function of the used GCN model, and the problem isn't bilevel. After training the GAM model, its output graph is used to train the GCN model, which is then used to augment the set of labelled nodes by considering its confident predictions as ground truth. This training process is repeated for k iterations, and is called co-training [4].

The weights between nodes computed by our $G2G$ bears similarity with attention mechanisms. After each GCN layer, the importance of edges is re-evaluated based on the similarity between nodes representation in that layer. The similarity criterion can be user-defined like the dot product [23, 35], learned locally at each layer by a single-layer feed-forward network [36], or a combination of both schemes [17]. In contrast to our method, these mechanisms are trained and used within a GCN model in one optimization problem, while the use of $G2G$ explicitly derive a graph of better quality as a by-product.

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 gradients of Eqs. (2a) and (2b).

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:

$$(\boldsymbol{A}_{\Theta})_{i,j} = \gamma\Big(\beta\big(\alpha(\boldsymbol{X}_i), \alpha(\boldsymbol{X}_j)\big), (\boldsymbol{A}_{obs})_{i,j}\Big) \ , \tag{6}$$

where:

- the encoder $\alpha : \mathbb{R}^p \to \mathbb{R}^{p_\alpha}$ is a Multi-Layer Perceptron MLP (see Appendix A for the model formula), that computes a new representation vector for a node in a new embedding space of dimension p_α.
- 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 merge 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:

$$\beta : (\tilde{\boldsymbol{X}}_i, \tilde{\boldsymbol{X}}_j) \mapsto (\tilde{\boldsymbol{X}}_i - \tilde{\boldsymbol{X}}_j)^2, \tag{7}$$

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

Algorithm 1 Learning algorithm

Input: $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{A}_{obs}$
Output: Θ : $G2G$ trained weights.
Hyperparameters: η_{out}, η_{in} (learning rates), τ_{in}, τ_{out} (number of iterations), q (number of unrolled inner iterations, in our work $q = 10$).
Randomly initialize Θ.
for $k = 1$ **to** τ_{out} **do**
  $\boldsymbol{A}_{\Theta} \leftarrow G2G(\boldsymbol{X}, \boldsymbol{A}_{obs})$
  Randomly initialize W_{Θ}.
  for $t = 1$ **to** τ_{in} **do**
    $\hat{\boldsymbol{Y}}_{W_{\Theta}}(\boldsymbol{X}, \boldsymbol{A}_{\Theta}) \leftarrow$ evaluate inner model.
    $\nabla_{W_{\Theta}} F_{in} \leftarrow$ *PyTorch* AD on $F_{in}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}_{W_{\Theta}}, \boldsymbol{A}_{\Theta})$
    $W_{\Theta} \leftarrow$ Adam-step$(W_{\Theta}, \nabla_{W_{\Theta}} F_{in}, \eta_{in})$
    if $t == \tau_{in} - q$ **then**
      Track next q inner updates as a function of Θ with *Higher* package.
    end if
  end for
  $\hat{\boldsymbol{Y}}_{W_{\Theta}}(\boldsymbol{X}, \boldsymbol{A}_{\Theta}) \leftarrow$ evaluate inner model.
  $\nabla_{\Theta} F_{out} \leftarrow$ *Higher* AD on $F_{out}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}_{W_{\Theta}}, \boldsymbol{A}_{\Theta})$
  $\Theta \leftarrow$ Adam-step$(\Theta, \nabla_{\Theta} F_{out}, \eta_{out})$
end for
Return: Θ, W_{Θ} (optional).

– 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 $[n] = \{1, \ldots, n\}$, and $\boldsymbol{P} \in \{0, 1\}^{n \times n}$ the according permutation matrix, then the following condition holds:

$$G2G(\boldsymbol{P}\boldsymbol{X}, \boldsymbol{P}\boldsymbol{A}_{obs}\boldsymbol{P}) = \boldsymbol{P}G2G(\boldsymbol{X}, \boldsymbol{A}_{obs})\boldsymbol{P} \ . \tag{8}$$

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 relabelling the points in the input dataset, and can generalize to new graphs.

3.2 Learning routine

Neither the minimizer of Eq. (2b) 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 replace W_{Θ} 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) as an optimizer [18], and we fix the number of iterations τ_{in}, the step size η_{in}, then we assume that W_Θ is close to $W_{\tau_{in},\Theta}$, with:

$$W_{t,\Theta} = W_{t-1,\Theta} - \eta_{in}\widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon) \ , \tag{9}$$

where $\widehat{m}_t = m_t/(1-\zeta_1^t)$, $\widehat{v}_t = v_t/(1-\zeta_2^t)$ are the bias-corrected moment estimates, ζ_1, ζ_2 are exponential decay rates defined by user (so is ϵ), $m_t = \zeta_1 m_{t-1} + (1-\zeta_1)\nabla_{W_{t-1,\Theta}} F_{in}$, $v_t = \zeta_2 v_{t-1} + (1-\zeta_2)(\nabla_{W_{t-1,\Theta}} F_{in})^2$ are the biased moment estimates with $m_0 = v_0 = 0$.

Although the output W_Θ is computed thanks to AD, which evaluates the gradient $\nabla_{W_{t,\Theta}} F_{in}$ for each update in Eq. (9), 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.,* $\boldsymbol{J}_{W_\Theta}(\Theta)$. That is to say, 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 $\nabla_\Theta F_{out}$. Furthermore, if we assume convergence in the inner problem (large value of τ_{in}), then $\boldsymbol{J}_{W_{\tau_{in},\Theta}}(W_{\tau_{in}-1,\Theta}) \approx 0$. Thus from the chain rule, it is sufficient to trace the last few updates, 10 in our case, to compute $\nabla_\Theta F_{out}$. This significantly saves memory and execution time compared to unrolling all τ_{in} updates.

Indeed, we use AD to get $\nabla_\Theta F_{out}$ and apply a gradient-based algorithm, Adam, to converge to a good set of weights Θ, as described in Algorithm 1.

4 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 points features. We also investigate the sensitivity of our framework *w.r.t.* the number of unrolled iterations. 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 each point uniformly at random from $[0,1]^p$ with $n = 256$, $p = 2$, unless otherwise specified. The ground-truth graph $\boldsymbol{A}^\star$ is then constructed s.t. an edge between points i, j has the weight $\exp\left(-\|\boldsymbol{X}'_i - \boldsymbol{X}'_j\|_2^2/2\sigma^2\right)$. We generate observed features $\boldsymbol{X}$ as a function of latent variables too $\boldsymbol{X} = f(\boldsymbol{X}') \in \mathbb{R}^{n\times p}$. Each point i in the inner training set V_{tr2} is labeled as follows:

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

where $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$ are randomly sampled from $[0,1]^p$, and r is a scaling factor such that labels lie in $[0,1]$. By this construction, the prior that the labelling function on the graph is smooth is met, and the Laplacian regularization can be applied as in Eq. (5). To generate labels in the outer training and the validation sets V_{tr1}, V_{val}, respectively, we plug the labels of V_{tr2} and $\boldsymbol{A}^\star$ in Eq. (5) with

$\lambda = 1$, and the solution holds the sought for labels. This way, if the $G2G$ model learns the ground-truth graph, the validation and outer losses equal to zero. To generate the observed graph $\boldsymbol{A}_{obs}$, we consider a classical model of random graphs:

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

Each dataset is evenly divided into 4 subsets: inner training, outer training, validation, and unlabelled 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 Laplacian regularization as in Eq. (5).

Real world datasets: we demonstrate the capacity of our method on common SSL benchmark datasets for node classification: Cora [22], CiteSeer [3], and PubMed [26]. 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 unlabelled ones *w.r.t.* their main topic. From the default train/validation/test split in [40, 19], we use the training set as the inner training set V_{tr2}, while we use half of the validation set as the outer training set V_{tr1}. The other half is kept as a validation set as in [11].

Models: our $G2G$ and GCN models are implemented using *PyTorch* [28] and *PyTorch* Geometric [8], respectively. The encoder β and the regressor γ in the $G2G$ model are an MLP of 1 hidden layer each. Likewise, the GCN has 1 hidden layer. All hidden layers in all models have 128 hidden neurons, and equipped with the $ReLu$ activation function. The GCN 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 GCN models, and $\eta_{out} = 0.003$. We use *Higher* package to track unrolled inner iterations and apply the second-order automatic differentiation to compute $\nabla_{\Theta} F_{out}$ [13]. We fix $\tau_{in} = 300$ with a GCN model and $\tau_{in} = 100$ with Laplacian regularization. We unroll the last 10 iterations, unless otherwise mentioned. GCN weights W_{Θ} and $\hat{\boldsymbol{Y}}$ when using the Laplacian regularization are initialized at random after each outer iteration, using 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.

4.1 *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: $f_0(x'_1, x'_2) =$

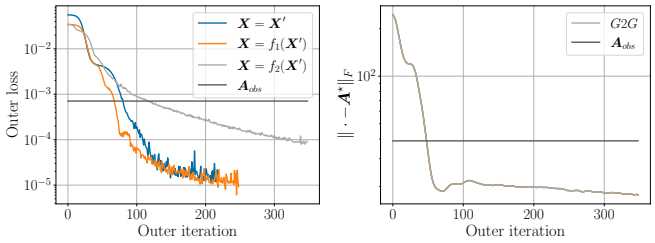

Fig. 1: The capacity of $G2G$ when varying $\boldsymbol{X}$ as a function of $\boldsymbol{X}'$. We try three functions as in Section 4.1. Left: we plot the outer loss. Right: we plot the error between the ground-truth graph $\boldsymbol{A}^\star$ and the $G2G$'s output in the case of f_2.

(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. 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. (5). 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 $\boldsymbol{A}^\star$, while being only guided by maximizing regression performance in Eq. (2a). This proves the power of the $G2G$ model, and the efficiency in computing gradients 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.

4.2 Sensitivity to the number of unrolled inner iterations

We observe the quality of AD-based gradients evaluated by *Higher* package for F_{out} as a function of the number of traced dynamics in the inner problem. We set $\tau_{in} = 800$ and vary the number of unrolled inner iterations q on the interval $[1, \tau_{in}]$, then we run our algorithm for each value. As seen in Fig. 2, *Higher* package successfully evaluates gradients when the number of unrolled updates goes from 1 up to 200, where we observe convergence and a good generalization on the validation set compared to $\boldsymbol{A}_{obs}$. Surprisingly, results degrade gradually with larger values and become unstable, which is not justified theoretically, as unrolling the few last or all inner updates at convergence should lead to the same Jacobian $\boldsymbol{J}_{W_\Theta}(\Theta)$. We conclude that *Higher* package becomes less accurate when it operates on large-scale computations. To verify this hypothesis, we designed a simpler bilevel problem and applied our algorithm. Our results show that *Higher* package indeed fails to track gradients with accelerated optimization methods, *e.g.,* Adam or SGD with momentum, as the number of unrolled inner iterations gets large. This problem is also reported in [11], but the source was not precisely

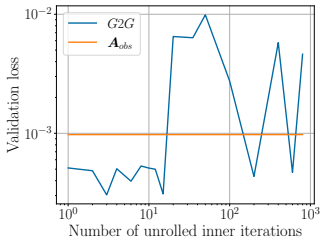

Fig. 2: Sensitivity to the number of unrolled inner iterations. We run the bilevel regime and plot the best validation loss for each value. $\tau_{in} = 800$.

detected to be the AD framework used in the according work. Based on this experiment, a good choice of this value would be from the range $[1, 200]$. To save memory and reduce training time, we choose to unroll $q = 10$ iterations.

4.3 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 GCN model in the second time. As seen in Table 1, $GCNs$ outperform the

Table 1: Benchmark against a GCN 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
GCN_{128}	77.0	67.0	75.2
$GCN_{128} + GAM$	86.2	73.5	86.0
$GCN_{128} + G2G$	79.6	71.8	79.2
$Laplacian + G2G$	78.9	54.7	70.3

Laplacian regularizer when used in the inner problem, which is expected due to the message passing model in $GCNs$. Besides, we see that our algorithm with a GCN in the inner problem yields significant improvement over the GCN 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.

5 Conclusion future work

One limitation in relational learning is that real-world graphs are noisy or not given. To solve this issue, we proposed to train a $G2G$ model to generate a better graph by solving a bilevel optimization. The inner optimization comprises training a traditional model, *e.g.,* GCN, on the output graph of $G2G$. The outer optimization uses the trained model to asses the performance of $G2G$ *w.r.t.* the outer objective function. This algorithm is the first variant of frameworks that train $G2G$ models using bilevel regimes, where such models has the advantage to generalize by expanding the graph on new points. To solve the bilevel problem, we replaced the inner problem by iterative applications of the Adam optimizer. Then, we used *Higher* package to perform second-order differentiation on Adam's output and compute the gradients to train the $G2G$ model. We showed empirically that applying *Higher* on the last 10 iterations is efficient to compute accurate gradients, which notably saves memory and training time. Experiments on synthetic data proves 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 given graphs, experiments on real datasets show that $G2G$ boosts the performance of GCN models and generalizes well on the validation set.

References

1. Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. Journal of Marchine Learning Research **18**, 1–43 (2018)
2. Bennett, K.P., Hu, J., Ji, X., Kunapuli, G., Pang, J.S.: Model selection via bilevel optimization. In: The 2006 IEEE International Joint Conference on Neural Network Proceedings. pp. 1922–1929. IEEE (2006)
3. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. ACM Trans. Knowl. Discov. Data **1**(1) (2007)
4. Blum, A., Mitchell, T.: Combining labeled and unlabeled data with co-training. In: Proceedings of the eleventh annual conference on Computational learning theory. pp. 92–100 (1998)
5. Bruna, J., Zaremba, W., Szlam, A., Lecun, Y.: Spectral networks and locally connected networks on graphs. In: International Conference on Learning Representations (ICLR2014), CBLS, April 2014 (2014)
6. Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., Murphy, K.: Machine learning on graphs: A model and comprehensive taxonomy. Journal of Machine Learning Research **23**(89), 1–64 (2022)
7. Colson, B., Marcotte, P., Savard, G.: An overview of bilevel optimization. Annals of operations research **153**(1), 235–256 (2007)
8. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)

9. Flamary, R., Rakotomamonjy, A., Gasso, G.: Learning constrained task similarities in graphregularized multi-task learning. Regularization, Optimization, Kernels, and Support Vector Machines **103** (2014)
10. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel programming for hyperparameter optimization and meta-learning. In: International Conference on Machine Learning. pp. 1568–1577. PMLR (2018)
11. Franceschi, L., Niepert, M., Pontil, M., He, X.: Learning discrete structures for graph neural networks. In: International conference on machine learning. pp. 1972–1982. PMLR (2019)
12. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: International conference on machine learning. pp. 1263–1272. PMLR (2017)
13. Grefenstette, E., Amos, B., Yarats, D., Htut, P.M., Molchanov, A., Meier, F., Kiela, D., Cho, K., Chintala, S.: Generalized inner loop meta-learning. arXiv preprint arXiv:1910.01727 (2019)
14. Ji, S., Pan, S., Cambria, E., Marttinen, P., Philip, S.Y.: A survey on knowledge graphs: Representation, acquisition, and applications. IEEE Transactions on Neural Networks and Learning Systems **33**(2), 494–514 (2021)
15. Kazemi, S.M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., Poupart, P.: Representation learning for dynamic graphs: A survey. J. Mach. Learn. Res. **21**(70), 1–73 (2020)
16. Keriven, N., Bietti, A., Vaiter, S.: Convergence and stability of graph convolutional networks on large random graphs. Advances in Neural Information Processing Systems **33**, 21512–21523 (2020)
17. Kim, D., Oh, A.: How to find your friendly neighborhood: Graph attention design with self-supervision. In: International Conference on Learning Representations (2021)
18. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (12 2014)
19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
20. Liben-Nowell, D., Kleinberg, J.: The link prediction problem for social networks. In: Proceedings of the twelfth international conference on Information and knowledge management. pp. 556–559 (2003)
21. Liu, Q., Allamanis, M., Brockschmidt, M., Gaunt, A.: Constrained graph variational autoencoders for molecule design. Advances in neural information processing systems **31** (2018)
22. Lu, Q., Getoor, L.: Link-based classification. In: ICML 2003 (2003)
23. Luong, T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 1412–1421. Association for Computational Linguistics, Lisbon, Portugal (Sep 2015)
24. Milanfar, P.: A tour of modern image filtering: New insights and methods, both practical and theoretical. IEEE signal processing magazine **30**(1), 106–128 (2012)
25. Morris, C., Ritzert, M., Fey, M., Hamilton, W.L., Lenssen, J.E., Rattan, G., Grohe, M.: Weisfeiler and leman go neural: Higher-order graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence. vol. 33, pp. 4602–4609 (2019)
26. Namata, G., London, B., Getoor, L., Huang, B., Edu, U.: Query-driven active surveying for collective classification. In: 10th International Workshop on Mining and Learning with Graphs. vol. 8, p. 1 (2012)

27. Pang, J., Cheung, G.: Graph laplacian regularization for image denoising: Analysis in the continuous domain. IEEE Transactions on Image Processing **26**(4), 1770–1785 (2017)
28. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)
29. Rong, Y., Bian, Y., Xu, T., Xie, W., Wei, Y., Huang, W., Huang, J.: Self-supervised graph transformer on large-scale molecular data. Advances in Neural Information Processing Systems **33**, 12559–12571 (2020)
30. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. science **290**(5500), 2323–2326 (2000)
31. Serviansky, H., Segol, N., Shlomi, J., Cranmer, K., Gross, E., Maron, H., Lipman, Y.: Set2graph: Learning graphs from sets. Advances in Neural Information Processing Systems **33**, 22080–22091 (2020)
32. Slepcev, D., Thorpe, M.: Analysis of p-laplacian regularization in semisupervised learning. SIAM Journal on Mathematical Analysis **51**(3), 2085–2120 (2019)
33. Stretcu, O., Viswanathan, K., Movshovitz-Attias, D., Platanios, E., Ravi, S., Tomkins, A.: Graph agreement models for semi-supervised learning. Advances in Neural Information Processing Systems **32** (2019)
34. Tenenbaum, J.B., Silva, V.d., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. science **290**(5500), 2319–2323 (2000)
35. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
36. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. International Conference on Learning Representations (2018)
37. Verma, A.: An introduction to automatic differentiation. Current Science pp. 804–807 (2000)
38. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems **32**(1), 4–24 (2020)
39. Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C.: Graph wavenet for deep spatial-temporal graph modeling. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. p. 1907–1913. IJCAI'19, AAAI Press (2019)
40. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48. p. 40–48. ICML'16, JMLR.org (2016)

A Models

We refer by MLP (Multi Layer Perceptron) to networks 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:

$$\boldsymbol{X}^{[l+1]} = \phi^{[l+1]}(\boldsymbol{X}^{[l]}\boldsymbol{W}_1^{[l+1]} + \boldsymbol{b}^{[l+1]}) \ , \tag{10}$$

where $W^{[l+1]} = \{\boldsymbol{W}_1^{[l+1]}, \boldsymbol{b}^{[l+1]}\}$ are the weights of this layer, and ϕ is the non-linear activation function. Likewise for a GCN model, the graph convolutional layer we consider computes $\boldsymbol{X}^{[l+1]}$ as follows [25]:

$$\boldsymbol{X}^{[l+1]} = \phi^{[l+1]}(\boldsymbol{X}^{[l]}\boldsymbol{W}_1^{[l+1]} + \boldsymbol{A}\boldsymbol{X}^{[l]}\boldsymbol{W}_2^{[l+1]} + \boldsymbol{b}^{[l+1]}) \ ,$$

where $\boldsymbol{A}$ is the input adjacency matrix, $W^{[l+1]} = \{\boldsymbol{W}_1^{[l+1]}, \boldsymbol{W}_2^{[l+1]}, \boldsymbol{b}^{[l+1]}\}$ are the weights of this layer, and ϕ is a non-linearity. ϕ is usually chosen to be the $ReLu$ function for hidden layers, and for the output layer either $softmax$ when solving a classification problem, or identity when solving a regression problem.

B Memory and computation costs

Tracking 10 inner iterations to construct gradients, chosen based on Section 4.2, our algorithm needs memory to place $G2G$ weights, 10 copies of the GCN weights, or 10 copies of the optimized labels $\hat{\boldsymbol{Y}}$ when using the Laplacian regularization method. That is in addition to points in the dataset, and other hyperparameters that can be ignored. Since GCN 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 [33], the graph attention models, and the LDS model. Regarding the computation cost, our algorithm and GAM have comparable costs when the number of co-training rounds equals the number of outer iterations. Whereas it costs less compared to the LDS method, thanks to tracking just the last 10 inner iterations by automatic differentiation. However, GAT models are still the least expensive among previous algorithms.