WHY PROPAGATE ALONE? PARALLEL USE OF LABELS AND FEATURES ON GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

One of the challenges of graph-based semi-supervised learning over ordinary supervised learning for classification tasks lies in label utilization. The direct use of ground-truth labels in graphs for training purposes can result in a parametric model learning trivial degenerate solutions (e.g., an identity mapping from input to output). In addressing this issue, a label trick has recently been proposed in the literature and applied to a wide range of graph neural network (GNN) architectures, achieving state-of-the-art results on various datasets. The essential idea is to randomly split the observed labels on the graph and use a fraction of them as input to the model (along with original node features), and predict the remaining fraction. Despite its success in enabling GNNs to propagate features and labels simultaneously, this approach has never been analyzed from a theoretical perspective, nor fully explored across certain natural use cases. In this paper, we demonstrate that under suitable settings, this stochastic trick can be reduced to a more interpretable deterministic form, allowing us to better explain its behavior, including an emergent regularization effect, and motivate broader application scenarios. Our experimental results corroborate these analyses while also demonstrating improved node classification performance applying the label trick in new domains.

1 INTRODUCTION

Node property prediction is a ubiquitous task involving graph data with node features and/or labels, with a wide range of instantiations across real-world scenarios such as node classification (Veličković et al., 2017) and link prediction, while also empowering graph classification (Gilmer et al., 2017), etc. Different from conventional machine learning problems where there is typically no explicit non-iid structure among samples, nodes are connected by edges, and a natural assumption is that labels and features vary smoothly over the graph. In order to effectively harness the information contained in node features and/or labels, this smoothing assumption has inspired two interrelated lines of research. First, graph neural networks (GNNs) (Kipf & Welling, 2016; Hamilton et al., 2017; Li et al., 2018; Xu et al., 2018b; Liao et al., 2019; Xu et al., 2018a) leverage a parameterized message passing strategy to convert the original node features into predictive embeddings that reflect the features of neighboring nodes. However, this approach does not directly utilize existing label information beyond their influence on model parameters through training. In contrast, label propagation algorithms (Zhu, 2005; Zhou et al., 2004; Zhang et al., 2006; Wang & Zhang, 2007; Karasuyama & Mamitsuka, 2013; Gong et al., 2016; Liu et al., 2018) spread label information via graph diffusion to make predictions, but cannot exploit node features.

As GNNs follow a similar propagation mechanism as the label propagation algorithm, the only principal difference being whether features or labels are smoothed across the graph, it is natural to consider combinations of the two for improving performance. Examples motivated by this intuition, at least to varying degrees, include APPNP (Klicpera et al., 2018), Correct and Smooth (C&S) (Huang et al., 2020), GCN-LPA (Wang & Leskovec, 2020), and TPN (Liu et al., 2018). While effective in many circumstances, these methods are not all end-to-end trainable and easily paired with arbitrary GNN architectures.

Among these various combination strategies, a recently emerged *label trick* has enjoyed widespread success in facilitating the parallel use of node features and labels via a stochastic label splitting technique. In brief, the basic idea is to use a randomly-selected potion of the training labels as

GNN inputs, concatenated with the original node features for making predictions on the remaining labels during each mini-batch. The ubiquity of this simple label trick is evidenced by its adoption across numerous GNN architectures and graph benchmarks (Sun & Wu, 2020; Wang et al., 2021; Kong et al., 2020; Li et al., 2021; Shi et al., 2020). And with respect to practical performance, this technique is foundational to many of the top-ranking submissions on the Open Graph Benchmark (OGB) leaderboard (Hu et al., 2020). For example, at the time of this submission, the top 10 results spanning multiple research teams all rely on the label trick, as do the top 3 results from the recent KDDCUP 2021 Large-Scale Challenge MAG240M-LSC (Hu et al., 2021).

And yet despite its far-reaching adoption, thus far the label trick has been motivated primarily as a training heuristic without a strong theoretical foundation. Moreover, many aspects of its underlying operational behavior have not been explored, with non-trivial open questions remaining. For example, while originally motivated from a stochastic perspective, is the label trick reducible to a more transparent deterministic form that is amenable to interpretation and analysis? Similarly, are there any indirect regularization effects with desirable (or possibly undesirable) downstream consequences? And how do the implicit predictions applied by the model to test nodes during the stochastic training process compare with the actual deterministic predictions used during inference? If there is a discrepancy, then the generalization ability of the model could be compromised. And finally, are there natural use cases for the label trick that have so far flown under the radar and been missed?

We take a step towards answering these questions via the following two primary contributions:

- We prove that in certain restricted settings, the original *stochastic* label trick can be reduced to an interpretable, *deterministic* training objective composed of two terms: (1) a data-fitting term that naturally resolves potential label leakage issues and maintains consistent predictions during training and inference, and (2) a regularization factor conditioned on graph structure that adapts to graph size and connectivity. Furthermore, complementary experiments applying the label trick to a broader class of graph neural network models corroborate that similar effects exists in more practical real-world settings, consistent with our theoretical findings.
- Although in prior work the label trick has already been integrated within a wide variety of GNN models, we introduce novel use-cases motivated by our analysis. This includes exploiting the label trick to: (i) train simple end-to-end variants of label propagation and C&S, and (ii) replace stochastic use cases of the label trick with more stable, deterministic analogues that are applicable to GNN models with linear propagation layers such as SGC (Wu et al., 2019), TWIRLS (Yang et al., 2021) and SIGN (Frasca et al., 2020). Empirical results on node classification benchmarks verify the efficacy of these simple enhancements.

Collectively, these efforts establish a more sturdy foundation for the label trick, and in doing so, help to ensure that it is not underutilized.

2 Background

Consider a graph $G = (V, E)$ with $n = |V|$ nodes, the node feature matrix is denoted by $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and the label matrix of the nodes is denoted by $\boldsymbol{Y} \in \mathbb{R}^{n \times c}$, with d and c being the number of channels of features and labels, respectively. Let $\boldsymbol{A}$ be the adjacency matrix, $\boldsymbol{D}$ the degree matrix and $\boldsymbol{S} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ the symmetric normalized adjacency matrix. The symmetric normalized Laplacian $\boldsymbol{L}$ can then be formulated as $\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{S}$. We also define a training mask matrix as $\boldsymbol{M}_{tr} = \begin{pmatrix} \boldsymbol{I}_m & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}_{n \times n}$, where w.l.o.g. we are assuming that the first m nodes, denoted $\mathcal{D}_{tr}$, form the training dataset. We use $\boldsymbol{P}$ to denote a *propagation matrix*, where the specific $\boldsymbol{P}$ will be described in each context.

2.1 Label Propagation Algorithm

Label propagation is a semi-supervised algorithm that predicts unlabeled nodes by propagating the observed labels across the edges of the graph, with the underlying smoothness assumption that two nodes connected by an edge are likely to share the same label. Following (Zhou et al., 2004; Yang et al., 2021), the implicit energy function of label propagation is given by

$$E(\boldsymbol{F}) = (1 - \lambda)\|\boldsymbol{F} - \boldsymbol{Y}_{tr}\|_2^2 + \lambda \operatorname{tr}[\boldsymbol{F}^\top \boldsymbol{L} \boldsymbol{F}], \tag{1}$$

where $\boldsymbol{Y}_{tr} = \boldsymbol{M}_{tr}\boldsymbol{Y}$ is label matrix of training nodes, and $\lambda \in (0, 1)$ is a regularization coefficient that determines the trade-off between the two terms. The first term is a *fitting constraint*, with the intuition that the predictions of a good classifier should remain close to the initial label assignments, while the second term introduces a *smoothness constraint*, which favors similar predictions between neighboring nodes in the graph.

It is not hard to derive the closed-formed optimal solution of the energy function $\boldsymbol{F}^* = \boldsymbol{P}\boldsymbol{Y}$, where $\boldsymbol{P} = (1-\lambda)(\boldsymbol{I}_n - \lambda\boldsymbol{S})^{-1}$. However, since the stated inverse is impractical to compute for large graphs, $\boldsymbol{P}\boldsymbol{Y}$ is often approximated in practice via $\boldsymbol{P} \approx (1-\lambda)\sum_{i=0}^{k}\lambda^i\boldsymbol{S}^i\boldsymbol{Y}$ such that $\boldsymbol{F}$ can be computed by the more efficient iterations $\boldsymbol{F}^{(k+1)} = \lambda\boldsymbol{S}\boldsymbol{F}^{(k)} + (1-\lambda)\boldsymbol{F}^{(0)}$, where $\boldsymbol{F}^{(0)} = \boldsymbol{Y}_{tr}$ and the propagation matrix $\boldsymbol{P}$ smooths the training labels across the edges in the graph.

2.2 Graph Neural Networks for Propagating Node Features

In contrast to the propagation of labels across the graph, GNN models transform and propagate node features using a series of feed-forward neural network layers. Popular examples include GCN (Kipf & Welling, 2016), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2017), and GIN (Xu et al., 2018a). For instance, the layer-wise propagation rule of GCN can be formulated as $\boldsymbol{X}^{(k+1)} = \sigma(\boldsymbol{S}\boldsymbol{X}^{(k)}\boldsymbol{W}^{(k)})$ where $\sigma(\cdot)$ is an activation function such as ReLU, $\boldsymbol{X}^{(k)}$ is the k-th layer node representations with $\boldsymbol{X}^{(0)} = \boldsymbol{X}$, and $\boldsymbol{W}^{(k)}$ is a trainable weight matrix of the k-th layer. Compared with the label propagation algorithm, GNNs can sometimes exhibit a more powerful generalization capability via the interaction between discriminative node features and trainable weights.

2.3 Combining Label and Feature Propagation

While performing satisfactorily in many circumstances, GNNs only indirectly incorporate ground-truth training labels via their influence on the learned model weights. But these labels are not actually used during inference, which can potentially degrade performance relative to label propagation, especially when the node features are noisy or unreliable. Therefore, it is natural to consider the combination of label *and* feature propagation to synergistically exploit the benefits of both as has been proposed in (Klicpera et al., 2018; Liu et al., 2018; Wang & Leskovec, 2020; Wang et al., 2021; Shi et al., 2020; Huang et al., 2020).

One of the most successful among these hybrid methods is the so-called *label trick*, which can be conveniently retrofitted within most standard GNN architectures while facilitating the parallel propagation of labels and features in an end-to-end trainable fashion. As mentioned previously, a number of top-performing GNN pipelines have already adopted this trick, which serves to establish its widespread relevance (Sun & Wu, 2020; Wang et al., 2021; Kong et al., 2020; Li et al., 2021; Shi et al., 2020) and motivates our investigation of its properties herein. To this end, we formally define the label trick as follows:

Definition 1 (label trick) *The label trick is based on creating random partitions of the training data as in $\mathcal{D}_{tr} = \mathcal{D}_{in} \cup \mathcal{D}_{out}$ and $\mathcal{D}_{in} \cap \mathcal{D}_{out} = \varnothing$, where node labels from $\mathcal{D}_{in}$ are concatenated with the original features $\boldsymbol{X}$ and provided as GNN inputs (for nodes not in $\mathcal{D}_{in}$ zero-padding is used), while the labels from $\mathcal{D}_{out}$ serve in the traditional role as supervision. The resulting training objective then becomes*

$$\mathop{\mathbb{E}}_{splits}\left[\sum_{i\in\mathcal{D}_{out}} \ell\left(\, \boldsymbol{y}_i,\ f\left(\boldsymbol{X}, \boldsymbol{Y}_{in}; \boldsymbol{\mathcal{W}}\right)_i \,\right)\right] \tag{2}$$

where $\boldsymbol{Y}_{in} \in \mathbb{R}^{n\times c}$, defined row-wise as $\boldsymbol{y}_{in,i} = \begin{cases}\boldsymbol{y}_i & \text{if } i \in \mathcal{D}_{in} \\ \boldsymbol{0} & \text{otherwise}\end{cases}$ for all i, the function $f(\boldsymbol{X}, \boldsymbol{Y}_{in}; \boldsymbol{\mathcal{W}})$ represents a message-passing neural network with parameters $\boldsymbol{\mathcal{W}}$ and the concatenation of $\boldsymbol{X}$ and $\boldsymbol{Y}_{in}$ as inputs, and $\ell(\cdot,\cdot)$ denotes a point-wise loss function over one sample/node. At inference time, we then use the predictor $f(\boldsymbol{X}, \boldsymbol{Y}_{tr}; \boldsymbol{\mathcal{W}})_i$ for all test nodes $i \notin \mathcal{D}_{tr}$.

3 RELIABLE RANDOMNESS THOUGH THE LABEL TRICK

Despite its widespread adoption, the label trick has thus far been motivated as merely a training heuristic without formal justification. To address this issue, we will now attempt to quantify the induced regularization effect that naturally emerges when using the label trick. However, since the formal analysis of deep networks is not generally feasible, we herein adopt the simplifying assumption that the function f from (2) is linear analogous to the SGC model from (Wu et al., 2019).

For simplicity of exposition, in Sections 3.1 and 3.2 we will consider the case where no node features are present to isolate label-trick-specific phenomena. Later in Section 3.3 we will reintroduce node features to present our general results.

3.1 LABEL TRICK WITHOUT NODE FEATURES

Assuming no node features $\boldsymbol{X}$, we begin by considering the deterministic node label loss

$$\mathcal{L}(\boldsymbol{W}) = \sum_{i \in \mathcal{D}_{tr}} \ell(\boldsymbol{y}_i, (\boldsymbol{P}\boldsymbol{Y}_{tr}\boldsymbol{W})_i), \tag{3}$$

where $(\ \cdot\ )_i$ indicates the i-th row of a matrix and $\boldsymbol{P}\boldsymbol{Y}_{tr}\boldsymbol{W}$ is a linear predictor akin to SGC, but with only the zero-padded training label matrix $\boldsymbol{Y}_{tr}$ as an input. However, directly employing (3) for training suffers from potential label leakage issues given that a simple identity mapping suffices to achieve the minimal loss at the expense of accurate generalization to test nodes. Furthermore, there exists an inherent asymmetry between the predictions computed for training nodes, where the corresponding labels are also used as inputs to the model, and the predictions for testing nodes where no labels are available.

At a conceptual level, these issues can be resolved by the label trick, in which case we introduce random splits of $\mathcal{D}_{tr}$ and modify (3) to

$$\mathcal{L}(\boldsymbol{W}) = \mathop{\mathbb{E}}_{splits}\Big[\sum_{i \in \mathcal{D}_{out}} \ell(\boldsymbol{y}_i, (\boldsymbol{P}\boldsymbol{Y}_{in}\boldsymbol{W})_i)\Big]. \tag{4}$$

For each random split, the resulting predictor only includes the label information of $\mathcal{D}_{in}$, and thus there is no unresolved label leakage issue when predicting the labels of $\mathcal{D}_{out}$.

In practice, we typically sample the splits by assigning a given node to $\mathcal{D}_{in}$ with some probability $\alpha \in (0, 1)$; otherwise the node is set to $\mathcal{D}_{out}$. It then follows that $\mathbb{E}[|\mathcal{D}_{in}|] = \alpha|\mathcal{D}_{tr}|$ and $\mathbb{E}[\boldsymbol{Y}_{in}] = \alpha\boldsymbol{Y}_{tr}$.

3.2 SELF-EXCLUDED SIMPLIFICATION OF THE LABEL TRICK

Because there exists an exponential number of different possible random splits, for analysis purposes (with later practical benefits as well) we first consider a simplified one-versus-all case whereby we enforce that $|\mathcal{D}_{out}| = 1$ across all random splits, with each node landing with equal probability in $\mathcal{D}_{out}$. In this situation, the objective function from (4) can be re-expressed more transparently without any expectation as

$$\begin{aligned}\mathcal{L}(\boldsymbol{W}) &= \mathop{\mathbb{E}}_{splits}\Big[\sum_{i \in \mathcal{D}_{out}} \ell(\boldsymbol{y}_i, [\boldsymbol{P}\boldsymbol{Y}_{in}\boldsymbol{W}]_i)\Big] = \mathop{\mathbb{E}}_{splits}\Big[\sum_{i \in \mathcal{D}_{out}} \ell\left(\boldsymbol{y}_i, [\boldsymbol{P}(\boldsymbol{Y}_{tr} - \boldsymbol{Y}_i)\boldsymbol{W}]_i\right)\Big] \\ &= \sum_{i \in \mathcal{D}_{tr}} \ell\left(\boldsymbol{y}_i, [(\boldsymbol{P} - \boldsymbol{C})\boldsymbol{Y}_{tr}\boldsymbol{W}]_i\right),\end{aligned} \tag{5}$$

where $\boldsymbol{Y}_i$ represents a matrix that shares the i-th row of $\boldsymbol{Y}$ and pads the rest with zeros, and $\boldsymbol{C} = \mathrm{diag}(\boldsymbol{P})$. This then motivates the revised predictor given by

$$f(\boldsymbol{Y}_{tr}; \boldsymbol{W}) = (\boldsymbol{P} - \boldsymbol{C})\boldsymbol{Y}_{tr}\boldsymbol{W}, \tag{6}$$

where $\boldsymbol{P}$ here can in principle be any reasonable propagation matrix, not necessarily the one associated with the original label propagation algorithm.

Remark 1 *From this expression we observe the intuitive role that $\boldsymbol{C}$ plays in blocking the direct pathway between each training node input label to the output predicted label for that same node.*

In this way the predictor propagates the labels of each training node excluding itself, and for both training and testing nodes alike, the predicted label of a node is only a function of other node labels. This resolves the asymmetry mentioned previously with respect to the predictions from (3).

Remark 2 *It is generally desirable that a candidate model produces the same predictions on test nodes during training and inference to better insure proper generalization. Fortunately, this is in fact the case when applying (6), which on test nodes makes the same predictions as label propagation. To see this, note that* $\boldsymbol{M}_{te}(\boldsymbol{P}-\boldsymbol{C})\boldsymbol{Y}_{tr} = \boldsymbol{M}_{te}\boldsymbol{P}\boldsymbol{Y}_{tr}$*, where* $\boldsymbol{M}_{te} = \boldsymbol{I}_n - \boldsymbol{M}_{tr}$ *is the diagonal mask matrix of test nodes.*

Although ultimately (6) will serve as a useful analysis tool below, it is also possible to adopt this predictor in certain practical settings. In this regard, $\boldsymbol{C}$ can be easily computed with the same computational complexity as is needed to approximate $\boldsymbol{P}$ as discussed in Section 2.1 (and for alternative propagation operators that are available explicitly, e.g., the normalized adjacency matrix, $\boldsymbol{C}$ is directly available).

3.3 Full Execution of the Label Trick as a Regularizer

We are now positioned to extend the self-excluded simplification of the label trick to full execution with arbitrary random sampling, as well as later, the reintroduction of node features. For this purpose, we first define $\boldsymbol{Y}_{out} = \boldsymbol{Y}_{tr} - \boldsymbol{Y}_{in}$, and also rescale by a factor of $1/\alpha$ to produce $\widetilde{\boldsymbol{Y}}_{in} = \boldsymbol{Y}_{in}/\alpha$. The latter allows us to maintain a consistent mean and variance of the predictor across different sampling probabilities.

Assuming a mean square error (MSE) loss as computed for each node via $\ell(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = ||\boldsymbol{y} - \widehat{\boldsymbol{y}}||_2^2$ (later we consider categorical cross-entropy), our overall objective is to minimize

$$\mathcal{L}(\boldsymbol{W}) = \mathop{\mathbb{E}}_{splits}\Big[\sum_{i\in\mathcal{D}_{out}} \ell(\boldsymbol{y}_i, (\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W})_i)\Big] = \mathop{\mathbb{E}}_{splits}\Big[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}\|_F^2\Big], \tag{7}$$

where $\boldsymbol{M}_{out}$ is a diagonal mask matrix defined such that $\boldsymbol{Y}_{out} = \boldsymbol{M}_{out}\boldsymbol{Y} = \boldsymbol{Y}_{tr} - \boldsymbol{Y}_{in}$ and the random splits follow a node-wise Bernoulli distribution with parameter α as discussed previously. We then have the following:

Theorem 1 *The label trick objective from (7) satisfies*

$$\frac{1}{1-\alpha}\mathop{\mathbb{E}}_{splits}\Big[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}\|_F^2\Big] = \|\boldsymbol{Y}_{tr} - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + \frac{1-\alpha}{\alpha}\|\boldsymbol{\Gamma}\boldsymbol{W}\|_F^2, \tag{8}$$

where $\boldsymbol{P}_{tr} = \boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{tr}$*,* $\boldsymbol{C}_{tr} = \mathrm{diag}(\boldsymbol{P}_{tr})$*, and* $\boldsymbol{\Gamma} = (\mathrm{diag}(\boldsymbol{P}_{tr}^2) - \boldsymbol{C}_{tr}^2)^{\frac{1}{2}}\boldsymbol{Y}_{tr}$*.*

This result implies that in the case of an MSE training loss and no node features, the label trick objective, with an arbitrary random splitting probability α, decomposes into: (i) the error accrued when using the self-excluded label propagation predictor from (6), and (ii) an additional quadratic regularization factor on the model weights that depends on α (more on this below). Furthermore, we can extend this analysis to include node features by incorporating the SGC-like linear predictor $\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x$ such that Theorem 1 can then naturally be generalized as follows:

Corollary 1 *Under the same conditions as Theorem 1, if we add the node feature term* $\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x$ *to the label-based predictor from (7), we have that*

$$\begin{aligned}&\frac{1}{1-\alpha}\mathop{\mathbb{E}}_{splits}\Big[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}_y\|_F^2\Big]\\ &= \|\boldsymbol{Y}_{tr} - \boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y\|_F^2 + \frac{1-\alpha}{\alpha}\|\boldsymbol{\Gamma}\boldsymbol{W}_y\|_F^2.\end{aligned} \tag{9}$$

The details of the proofs of Theorem 1 and Corollary 1 are provided in Appendix A.1. This then effectively leads to the more general, feature and label aware predictor

$$f(\boldsymbol{X}, \boldsymbol{Y}_{tr}; \boldsymbol{\mathcal{W}}) = \boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x + (\boldsymbol{P} - \boldsymbol{C})\boldsymbol{Y}_{tr}\boldsymbol{W}_y, \tag{10}$$

where $\boldsymbol{\mathcal{W}} = \{\boldsymbol{W}_x, \boldsymbol{W}_y\}$. These theoretical results reveal a number of interesting properties regarding how the label trick behaves, which we summarize as follows:

Remark 3 *Although the original loss involves an expectation over random data splits that is somewhat difficult to interpret, based on (9) the label trick can be interpreted as a deterministic data-fitting term involving both labels and features that mitigates label leakage, and an adaptive, graph-dependent regularization factor applied only to the label weights. Moreover, we can easily verify from (10) that the model effectively applies the same prediction to test nodes during both training and inference, consistent with Remark 2.*

Remark 4 *Regarding the $\boldsymbol{\Gamma}$-dependent penalty term, if the graph has no edges, then there is no chance for overfitting to labels and $(\text{diag}(\boldsymbol{P}_{tr}^2) - \boldsymbol{C}_{tr}^2)^{\frac{1}{2}} = 0$ shuts off the regularization. In contrast, for a fully connected graph, the value of $\boldsymbol{\Gamma}$ can be significantly larger, which can potentially provide a beneficial regularization effect. Additionally, given that $\boldsymbol{\Gamma}$ also grows larger with graph size (assuming edges grow as well), $\|\boldsymbol{\Gamma}\boldsymbol{W}_y\|_F^2$ scales proportionately with the data fitting term, which is generally expected to increase linearly with the number of nodes. Hence (9) is naturally balanced to problem size.*

Remark 5 *The splitting probability α in (9) controls the regularization strength. Specifically, when α tends to zero, fewer labels are used as input to predict a large number of output labels, which may be less reliable and corresponds with adding a larger regularization term. Additionally, it means placing more emphasis on the original node features and downplaying the importance of the labels as input in (9), which explains the addition of a penalty on $\boldsymbol{W}_y$. Conversely, when α tends to one, more labels are used as input to predict the output. Specifically, when $\boldsymbol{M}_{out} \neq 0$ the loss closely approximates the self-excluded label trick, while when $\boldsymbol{M}_{out} = 0$ the contribution to the expectation is zero and therefore does not influence the loss. So this situation naturally corresponds with canceling out the regularization term.*

Remark 6 *The regularization term is also loosely analogous to the one associated with dropout (Srivastava et al., 2014) for the case of linear regression, where the splitting probability α is similar to the keep probability of dropout. A smaller α means that fewer labels are used as input, implying stronger regularization. While this is an interesting association, there remain critical differences, such as the natural emergence of $\boldsymbol{C}$ which complicates the problem considerably; see the proof for further details.*

We now turn to the categorical cross-entropy loss, which is more commonly applied to node classification problems. While we can no longer compute closed-form simplifications as we could with MSE, it is nonetheless possible to show that the resulting objective when using the original label trick is an upper bound on the analogous objective from the self-excluded label trick. More specifically, we have the following (see Appendix A.3 for the proof):

Theorem 2 *Under the same conditions as in Theorem 1, if we replace the MSE loss with categorical cross-entropy we obtain the bound*

$$\begin{aligned} &\frac{1}{1-\alpha} \mathop{\mathbb{E}}_{splits} \left[\text{CrossEntropy}_{\mathcal{D}_{out}}(\boldsymbol{Y}_{out}, \boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x + \boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}_y) \right] \\ &\geq \text{CrossEntropy}_{\mathcal{D}_{tr}}(\boldsymbol{Y}_{tr}, \boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x + (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y), \end{aligned} \tag{11}$$

where $\text{CrossEntropy}_S(\cdot, \cdot)$ *denotes the sum of row-wise cross-entropy of S.*

4 Broader Use Cases of the Label Trick

Although the label trick has already been integrated within a wide variety of GNN pipelines, in this section we introduce three novel use-cases motivated by our analysis.

4.1 Trainable Label Propagation

In Sections 3.1 and 3.2 we excluded the use of node features to simplify the exposition of the label trick; however, analytical points aside, the presented methodology can also be useful in and of itself for facilitating a simple, trainable label propagation baseline.

The original label propagation algorithm from (Zhou et al., 2004) is motivated as a parameter-free, deterministic mapping from a training label set to predictions across the entire graph. However,

clearly the randomized label trick from 3.1, or its deterministic simplification from Section 3.2 can be adopted to learn a label propagation weight matrix $\boldsymbol{W}$. The latter represents a reasonable enhancement that can potentially help to compensate for interrelated class labels that may arise in multi-label settings. In contrast, the original label propagation algorithm implicitly assumes that different classes are independent. Beyond this, other entry points for adding trainable weights are also feasible such as node-dependent weights, nonlinear weighted mappings, step-wise weights for heterophily graphs, or weights for different node types for heterogeneous graphs.

4.2 Trainable Correct and Smooth

Correct and Smooth (C&S) is a simple yet powerful method which consists of multiple stages. A prediction matrix $\tilde{\boldsymbol{Y}}$ is first obtained whose rows correspond with a prediction from a shallow node-wise model. $\tilde{\boldsymbol{Y}}$ is subsequently modified via two post-processing steps, *Correct* and *Smooth*, using two propagation matrices $\boldsymbol{P}_c, \boldsymbol{P}_s$, where typically $\boldsymbol{P}_i = (1-\lambda_i)(\boldsymbol{I}_n - \lambda_i \boldsymbol{S})^{-1}, i \in \{c, s\}$. Namely, we define the difference between the ground truth and predictions on the training set $\boldsymbol{E} = \boldsymbol{Y}_{tr} - \tilde{\boldsymbol{Y}}_{tr}$ as the *error matrix*, and $\tilde{\boldsymbol{E}} = \gamma(\boldsymbol{P}_c \boldsymbol{E})$ as the *correction matrix*, where γ is some row-independent scaling function. The smoothed prediction is $f_{C\&S}(\tilde{\boldsymbol{Y}}) = \boldsymbol{P}_s(\boldsymbol{Y}_{tr} + \boldsymbol{M}_{te}(\tilde{\boldsymbol{Y}} + \tilde{\boldsymbol{E}}))$.

With the label trick, we can equip C&S with trainable weights to further boost performance. To this end, we first split the training dataset into $\boldsymbol{Y}_{out}$ and $\boldsymbol{Y}_{in}$ as before. Then we multiply $\tilde{\boldsymbol{E}}_{in} = \gamma(\boldsymbol{P}_c(\boldsymbol{Y}_{in} - \tilde{\boldsymbol{Y}}_{in}))$, the correction matrix with respect to $\boldsymbol{Y}_{in}$, with a weight matrix $\boldsymbol{W}_c$. We also multiply the result after smoothing with another weight matrix $\boldsymbol{W}_s$. Thus the overall predictor is

$$\begin{aligned} f_{TC\&S}(\tilde{\boldsymbol{Y}}; \boldsymbol{\mathcal{W}}) &= \boldsymbol{P}_s(\boldsymbol{Y}_{in} + (\boldsymbol{M}_{te} + \boldsymbol{M}_{out})(\tilde{\boldsymbol{Y}} + \tilde{\boldsymbol{E}}_{in}\boldsymbol{W}_c))\boldsymbol{W}_s \\ &= \boldsymbol{P}_s((\boldsymbol{Y}_{in} + (\boldsymbol{M}_{te} + \boldsymbol{M}_{out})\tilde{\boldsymbol{Y}})\boldsymbol{W}_s + \boldsymbol{P}_s(\boldsymbol{M}_{te} + \boldsymbol{M}_{out})\tilde{\boldsymbol{E}}_{in}\boldsymbol{W}_c\boldsymbol{W}_s \\ &= \boldsymbol{P}_s((\boldsymbol{Y}_{in} + (\boldsymbol{M}_{te} + \boldsymbol{M}_{out})\tilde{\boldsymbol{Y}})\boldsymbol{W}_s + \boldsymbol{P}_s(\boldsymbol{M}_{te} + \boldsymbol{M}_{out})\tilde{\boldsymbol{E}}_{in}\hat{\boldsymbol{W}}_c, \end{aligned} \tag{12}$$

where $\boldsymbol{\mathcal{W}} = \{\hat{\boldsymbol{W}}_c, \boldsymbol{W}_s\}$ and $\hat{\boldsymbol{W}}_c$ is the reparameterization of $\boldsymbol{W}_c\boldsymbol{W}_s$.

Then the objective function for optimizing $\boldsymbol{\mathcal{W}}$ is

$$\mathcal{L}(\boldsymbol{W}) = \mathop{\mathbb{E}}_{splits}\Big[\sum_{i \in \mathcal{D}_{out}} \ell(\boldsymbol{y}_i, f_{TC\&S}(\boldsymbol{Y}_{in}; \boldsymbol{\mathcal{W}})_i)\Big]. \tag{13}$$

Since this objective solves the label leakage issue, it allows end-to-end training of C&S with gradients passing through both neural network layers and C&S steps. At times, however, this approach may have disadvantages, including potential overfitting problems or inefficiencies due to computationally expensive backpropagation, and alternative approach is to preserve the two-stage training. In this case, the base prediction in the first stage is the same as the original algorithm, while we just train C&S with parameters as post-processing steps.

4.3 Deterministic Application to GNNs with Linear Propagation Layers

The prevailing GNN methods follow the architecture of message passing neural networks (MPNNs). Among these GNN methods are variants that share node embeddings only through linear propagation layers. Representative examples include SGC (Wu et al., 2019), SIGN (Frasca et al., 2020) and TWIRLS (Yang et al., 2021). We then show how to apply the deterministic label trick algorithm as introduced in Section 3.2 with the aforementioned GNN methods.

We begin with a linear SGC model. In this case, we can compute $(\boldsymbol{P} - \boldsymbol{C})\boldsymbol{Y}_{tr}$ beforehand as the self-excluded label information and then train the resulting features individually without graph information, while avoiding label leakage problems. If desired, we can also concatenate with the original node features. In this way, we have an algorithm that minimizes an energy function involving both labels and input features.

Additionally, for more complex situations where the propagation layers are not at the beginning of the model, the predictor can be more complicated such as $f(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\mathcal{W}}) =$

$$h_1\Big(\sum_i (\boldsymbol{\mathcal{P}} h_0([\boldsymbol{X}, \boldsymbol{Y} - \boldsymbol{Y}_i]))_i\Big) = h_1(\boldsymbol{\mathcal{P}} h_0([\boldsymbol{X}, \boldsymbol{Y}]) - \boldsymbol{\mathcal{C}} h_0([\boldsymbol{X}, \boldsymbol{Y}]) + \boldsymbol{\mathcal{C}} h_0([\boldsymbol{X}, \boldsymbol{0}])), \tag{14}$$

where $[\cdot, \cdot]$ denotes the concatenation operation, $\boldsymbol{\mathcal{P}} = [\boldsymbol{P}_0, \boldsymbol{P}_1, \ldots, \boldsymbol{P}_{k-1}]^T$ is the integrated propagation matrix, $\boldsymbol{\mathcal{C}} = [\mathrm{diag}(\boldsymbol{P}_0), \mathrm{diag}(\boldsymbol{P}_1), \ldots, \mathrm{diag}(\boldsymbol{P}_{k-1})]^T$, h_0 and h_1 can be arbitrary node-independent functions, typically multi-layer perceptrons (MLPs).

5 EXPERIMENTS

As mentioned previously, the effectiveness of the label trick in improving GNN performance has already been demonstrated in prior work, and therefore, our goal here is not to repeat these efforts. Instead, in this section we will focus on conducting experiments that complement our analysis from Section 3 and showcase the broader application scenarios from Section 4.

We use four relatively large datasets for evaluation, namely Cora-full, Pubmed (Sen et al., 2008), ogbn-arxiv and ogbn-products (Hu et al., 2020). For Cora-full and Pubmed, we randomly split the nodes into training, validation, and test datasets with the ratio of 6:2:2 using different random seeds. For ogbn-arxiv and ogbn-products, we adopt the standard split from OGB (Hu et al., 2020). We report the average classification accuracy or mean square error (MSE) and standard deviation after 10 runs with different random seeds, and these are the results on the test dataset when not otherwise specified.

In addition, the label trick can actually be implemented in two ways. The first is the one that randomly splits the training nodes, and the second is the simpler version introduced herein with the deterministic one-versus-all splitting strategy, which does not require any random sampling. To differentiate the two versions, we denote the stochastic label trick with random splits by **label trick (S)**, and the deterministic one by **label trick (D)**. We find that sometimes it can be advantageous to have a lower splitting probability α, but this will make the training process slower. One way to approximate it is to use the deterministic version, which performs more efficiently. We conduct experiments with both, thus explaining our analysis with a comparison performed by the two versions.

5.1 TRAINABLE LABEL PROPAGATION

We first investigate the performance of applying the label trick to label propagation as introduced in (2) in the absence of features and compare it with the original label propagation algorithm. Table 1 shows that the trainable weights applied to the label propagation algorithm can boost the performance consistently. Given the notable simplicity of label propagation, this represents a useful enhancement.

Table 1. Accuracy results (%) of label propagation and trainable label propagation.

Method	Label Propagation	Trainable Label Propagation
Cora-full	66.44 ± 0.93	**67.23 ± 0.66**
Pubmed	83.45 ± 0.63	**83.52 ± 0.59**
Arxiv	67.11 ± 0.00	**68.42 ± 0.01**
Products	74.24 ± 0.00	**75.61 ± 0.21**

Table 2. MSE on node regression benchmarks with tabular features.

Method	C&S	Trainable C&S
House	0.51 ± 0.01	**0.45 ± 0.01**
County	1.42 ± 0.14	**1.13 ± 0.09**
VK	7.02 ± 0.20	**6.95 ± 0.22**
Avazu	**0.106 ± 0.014**	**0.106 ± 0.014**

5.2 DETERMINISTIC LABEL TRICK APPLIED TO GNNS WITH LINEAR LAYERS

We also test the deterministic label trick by applying it to different GNN architectures involving linear propagation layers along with (14). The results are in Table 3.

Table 3. Accuracy results (%) with/without the label trick (D).

Method	SGC		TWIRLS		SIGN	
label trick (D)	✗	✓	✗	✓	✗	✓
Cora-full	**65.87 ± 0.61**	65.81 ± 0.69	70.36 ± 0.51	**70.40 ± 0.71**	**68.54 ± 0.76**	68.44 ± 0.88
Pubmed	85.02 ± 0.43	**85.23 ± 0.57**	89.81 ± 0.56	**90.08 ± 0.52**	87.94 ± 0.52	**88.09 ± 0.59**
Arxiv	69.07 ± 0.01	**70.22 ± 0.03**	72.93 ± 0.19	**73.22 ± 0.10**	69.97 ± 0.16	**70.98 ± 0.21**

From these results we observe that on Pubmed and ogbn-arxiv, the deterministic label trick boosts the performance consistently on different models, while on Cora-full, it performs comparably. This is reasonable given that the training accuracy on Cora-full (not shown) is close to 100%, in which case the model does not benefit much from the information from ground-truth training labels.

5.3 EFFECT OF THE SPLITTING PROBABILITY

In terms of the effect of different splitting probabilities, we compare the accuracy results of a linear model and GNN on ogbn-arxiv as shown in Figure 1. For the linear model, α serves as a regularization coefficient. More specifically, when α tends to zero, the model converges to one without the label

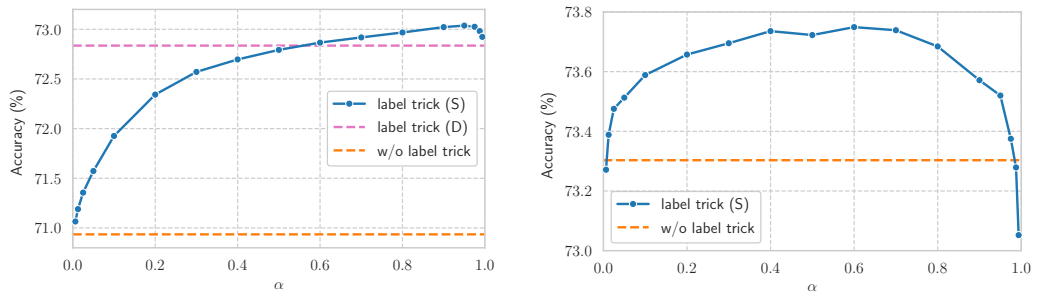

Figure 1. Accuracy results on validation dataset of linear propagation of features and labels (left) and GNNs (right) varying α.

trick; while when α tends to one, it converges to the case with self-excluded label trick. For the GNN, α has a similar effect as predicted by theory for the linear models. As α decreases, the model converges to that without the label trick. However, when α tends to one, the performance decreases in a more pronounced fashion, likely due to the similar effect of a small batch size of stochastic gradient descent applied to nonlinear neural networks. Moreover, a larger α is preferable for linear models in contrast to GNNs, probably because a simple model does not require strong regularization.

5.4 Trainable Correct and Smooth

We also verified the effectiveness of our approach when applied to Correct and Smooth (C&S), as described in Section 4.2. Due to the significant impact of end-to-end training on the model, it is not suitable for direct comparison with vanilla C&S for a natural ablation. Therefore, in this experiment, we train the C&S as post-processing steps. In Table 4, we report the test and validation accuracy, showing that our method outperforms the vanilla C&S on Cora-full and Pubmed. And for ogbn-arxiv and ogbn-products, the trainable C&S performs better in terms of validation accuracy while is comparable to vanilla C&S in terms of test accuracy.

Table 4. Test and validation accuracy (%) of C&S and trainable C&S. Validation accuracy reported in parentheses.

Method	MLP	MLP+C&S	MLP+Trainable C&S
Cora-full	60.12 ± 0.29 (61.09 ± 0.39)	66.95 ± 1.46 (68.26 ± 1.24)	**67.89 ± 1.37** (**69.09 ± 1.25**)
Pubmed	88.72 ± 0.34 (89.25 ± 0.26)	89.12 ± 0.27 (89.45 ± 0.17)	**89.76 ± 0.17** (**89.62 ± 0.18**)
Arxiv	71.48 ± 0.15 (72.95 ± 0.05)	**73.05 ± 0.35** (74.01 ± 0.17)	73.03 ± 0.18 (**74.44 ± 0.08**)
Products	67.60 ± 0.15 (87.07 ± 0.05)	**83.16 ± 0.13** (91.70 ± 0.06)	83.10 ± 0.15 (**91.99 ± 0.07**)

In principle, C&S can be applied to any base predictor model. To accommodate tabular node features, we choose to use gradient boosted decision trees, which can be trained end-to-end using methods such as (Authors, 2021; Ivanov & Prokhorenkova, 2021) combined with the label trick to avoid data leakage issues as we have discussed. For evaluation, we adopt the four tabular node regression data sets from (Ivanov & Prokhorenkova, 2021) and train using the approach from (Authors, 2021). Results are shown in Table 2, where it is clear that the label trick can significantly improve performance.

6 Conclusion

In this work we closely examine from a theoretical prospective the recently emerged label trick, which enables the parallel propagation of labels and features and benefits various SOTA GNN architectures, and yet has thus far not be subject to rigorous analysis. In filling up this gap, we first introduce self-excluded simplification of the label trick by applying an extreme splitting strategy, and then prove that the random splits of the label trick can be regarded as introducing a regularization effect on the self-excluded label weights. Besides the success of this label trick with random splits on GNNs, we also discuss broader applications in the following two aspects: (1) empowering other graph-based methods with trainable weights, (2) getting rid of effect of randomness by incorporating self-excluded propagation within GNNs composed of linear propagation layers. We verify these applications and evaluate the performance gains against existing approaches over multiple benchmark datasets.

REFERENCES

Anomymous Authors. *under review*, 2021. 9

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020. 2, 7

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017. 1

Chen Gong, Dacheng Tao, Wei Liu, Liu Liu, and Jie Yang. Label propagation via teaching-to-learn and learning-to-teach. *IEEE transactions on neural networks and learning systems*, 28(6): 1452–1465, 2016. 1

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017. 1, 3

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *arXiv:2005.00687*, 2020. 2, 8

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. In *arXiv:2103.09430*, 2021. 2

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. Combining label propagation and simple models out-performs graph neural networks. In *arXiv:2010.13993*, 2020. 1, 3, 17

Sergei Ivanov and Liudmila Prokhorenkova. Boost then convolve: Gradient boosting meets graph neural networks. *arXiv preprint arXiv:2101.08543*, 2021. 9

Masayuki Karasuyama and Hiroshi Mamitsuka. Manifold-based similarity adaptation for label propagation. *NIPS*, 2013. 1

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1, 3

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *arXiv:1810.05997*, 2018. 1, 3

Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. In *arXiv:2010.09891*, 2020. 2, 3

Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. *arXiv preprint arXiv:2106.07476*, 2021. 2, 3

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018. 1

Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*, 2019. 1

Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. *arXiv preprint arXiv:1805.10002*, 2018. 1, 3

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. In *AI magazine*, 2008. 8

Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In *arXiv:2009.03509*, 2020. 2, 3

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014. 6

Chuxiong Sun and Guoshi Wu. Adaptive graph diffusion networks with hop-wise attention. In *arXiv:2012.15024*, 2020. 2, 3

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 1, 3

Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, 2007. 1

Hongwei Wang and Jure Leskovec. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755*, 2020. 1, 3

Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf. Bag of tricks for node classification with graph neural networks. *arXiv preprint arXiv:2103.13355*, 2021. 2, 3

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019. 2, 4, 7

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018a. 1, 3

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018b. 1

Yongyi Yang, Tang Liu, Yangkun Wang, Jinjing Zhou, Quan Gan, Zhewei Wei, Zheng Zhang, Zengfeng Huang, and David Wipf. Graph neural networks inspired by classical iterative algorithms. In *ICML*, 2021. 2, 7, 16

Xinhua Zhang, Wee Sun Lee, et al. Hyperparameter learning for graph based semi-supervised learning algorithms. In *NIPS*, volume 33, 2006. 1

Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *NIPS*, 2004. 1, 2, 6

Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005. 1

A PROOFS

A.1 PROOF OF THEOREM 1

We first define the mask matrix for input labels:

$$\boldsymbol{M}_{in} = \begin{pmatrix} \mathrm{diag}(r_i) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, r_i \sim \text{Bernoulli}(\alpha), \tag{15}$$

where $\alpha \in (0, 1)$ is the label rate, and the mask matrix for output is defined as $\boldsymbol{M}_{out} = \boldsymbol{M}_{tr} - \boldsymbol{M}_{in}$.

Then the objective is

$$\begin{aligned}
\mathcal{L}(\boldsymbol{W}) &= \mathop{\mathbb{E}}_{splits}[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}\|_F^2] \\
&= \mathop{\mathbb{E}}_{splits}\left[\left\|\boldsymbol{Y}_{out} - \frac{1}{\alpha}\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{Y}_{in}\boldsymbol{W}\right\|_F^2\right] \\
&= \mathop{\mathbb{E}}_{splits}[\|\boldsymbol{Y}_{out}\|_F^2] - \frac{2}{\alpha}\mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{Y}_{out}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{Y}_{in}\boldsymbol{W}]] + \frac{1}{\alpha^2}\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{Y}_{in}\boldsymbol{W}\|_F^2] \\
&= \mathop{\mathbb{E}}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{Y}_{tr}\|_F^2] - \frac{2}{\alpha}\mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}]] + \frac{1}{\alpha^2}\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2].
\end{aligned} \tag{16}$$

Next we compute each term separately.

For notational convenience, we denote the i-th row and j-th column entry of $\boldsymbol{M}_{in}$ by M_{ij}.

1. For $\mathbb{E}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{Y}_{tr}\|_F^2]$,
   we have

$$\begin{aligned}
\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{Y}_{tr}\|_F^2] &= \mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{out}^T\boldsymbol{M}_{out}\boldsymbol{Y}_{tr}]] \\
&= \mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{out}\boldsymbol{Y}_{tr}]] \\
&= \mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{M}_{out}\boldsymbol{Y}_{tr}\boldsymbol{Y}_{tr}^T]] \\
&= \mathrm{tr}[\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{out}]\boldsymbol{Y}_{tr}\boldsymbol{Y}_{tr}^T] \\
&= (1-\alpha)\,\mathrm{tr}[\boldsymbol{M}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{Y}_{tr}^T] \\
&= (1-\alpha)\,\mathrm{tr}[\boldsymbol{Y}_{tr}\boldsymbol{Y}_{tr}^T] \\
&= (1-\alpha)\|\boldsymbol{Y}_{tr}\|_F^2.
\end{aligned} \tag{17}$$

2. For $\mathbb{E}_{splits}[\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}]]$,
   since $\boldsymbol{M}_{out} = \boldsymbol{M}_{tr} - \boldsymbol{M}_{in}$,

$$\begin{aligned}
\mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}]] &= \mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T]] \\
&= \mathrm{tr}[\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T] \\
&= \mathrm{tr}[\mathop{\mathbb{E}}_{splits}[(\boldsymbol{M}_{tr} - \boldsymbol{M}_{in})\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T] \\
&= \mathrm{tr}[(\boldsymbol{M}_{tr}\boldsymbol{P}\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}] - \mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}])\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T] \\
&= \mathrm{tr}[(\alpha\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{tr} - \mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}])\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T] \\
&= \alpha\,\mathrm{tr}[\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T] - \mathrm{tr}\left[\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T\right] \\
&= \alpha\,\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{P}\boldsymbol{Y}_{tr}\boldsymbol{W}] - \mathrm{tr}\left[\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T\right]
\end{aligned} \tag{18}$$

Let us calculate $\mathbb{E}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]$.
Since $\boldsymbol{M}_{in}$ is a diagonal matrix,

$$\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}] = (\mathop{\mathbb{E}}_{splits}[M_{ii}M_{jj}P_{ij}]) = \alpha^2\boldsymbol{P}_{tr} + (\alpha-\alpha^2)\boldsymbol{C}_{tr}, \tag{19}$$

where $\boldsymbol{P}_{tr} = \boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{tr}$ and $\boldsymbol{C}_{tr} = \mathrm{diag}(\boldsymbol{P}_{tr})$.
Then we have

$$\mathrm{tr}[\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}\boldsymbol{Y}_{tr}^T] = \alpha^2\,\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}] + (\alpha-\alpha^2)\,\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}]. \tag{20}$$

Therefore,

$$\begin{aligned}\mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}]] &= (\alpha-\alpha^2)\,\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}] - (\alpha-\alpha^2)\,\mathrm{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}] \\ &= (\alpha-\alpha^2)\,\mathrm{tr}[\boldsymbol{Y}_{tr}^T(\boldsymbol{P}_{tr}-\boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}].\end{aligned} \tag{21}$$

3. For $\mathbb{E}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2]$,
we have

$$\begin{aligned}&\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2] \\ &= \mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{out}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}]] \\ &= \mathop{\mathbb{E}}_{splits}[\mathrm{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}]] \\ &= \mathrm{tr}[\mathop{\mathbb{E}}_{splits}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{in}^T\boldsymbol{P}^T(\boldsymbol{M}_{tr}-\boldsymbol{M}_{in})\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}] \\ &= \mathrm{tr}[\mathop{\mathbb{E}}_{splits}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}] - \mathrm{tr}[\mathop{\mathbb{E}}_{splits}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}] \\ &= \mathrm{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}] - \mathrm{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}].\end{aligned} \tag{22}$$

Let us consider each term separately.

(a) To compute $\mathrm{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\,\mathbb{E}_{splits}[\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}]$,
we first define $\boldsymbol{Q}_{tr} = \mathrm{diag}(\boldsymbol{P}_{tr}^2)^{\frac{1}{2}}$, then

$$\begin{aligned}\mathop{\mathbb{E}}_{splits}\left[\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{in}\right] &= \mathop{\mathbb{E}}_{splits}\left[\sum_{k=1}^{m}M_{ii}M_{jj}P_{ki}P_{kj}\right] \\ &= \mathop{\mathbb{E}}_{splits}[M_{ii}M_{jj}]\sum_{k=1}^{m}P_{ki}P_{kj} \\ &= \alpha^2P_{tr}^T\boldsymbol{P}_{tr} + (\alpha-\alpha^2)\,\mathrm{diag}(\boldsymbol{P}_{tr}^2) \\ &= \alpha^2P_{tr}^T\boldsymbol{P}_{tr} + (\alpha-\alpha^2)(\boldsymbol{Q}_{tr}^2).\end{aligned} \tag{23}$$

Therefore,

$$\mathrm{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}] = \alpha^2\|\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + (\alpha-\alpha^2)\|\boldsymbol{Q}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2. \tag{24}$$

(b) To compute $\mathrm{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\,\mathbb{E}_{splits}[\boldsymbol{M}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}]$,
we have that

$$\begin{aligned}\left(\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}^T\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\right)_{ij} &= \left(\mathop{\mathbb{E}}_{splits}[M_{ii}M_{jj}\sum_k M_{kk}P_{ki}P_{kj}]\right)_{ij} \\ &= \left(\sum_k\mathop{\mathbb{E}}_{splits}[M_{ii}M_{jj}M_{kk}]P_{ki}P_{kj}\right)_{ij}.\end{aligned} \tag{25}$$

- For the diagonal entries, where $i = j$, we have

$$\sum_{k=1}^{n} \mathop{\mathbb{E}}_{splits}[M_{ii}M_{jj}M_{kk}]P_{ki}P_{kj} = \sum_{k} \alpha^2 P_{tr,ki}^2 + (\alpha - \alpha^2)P_{tr,ii}^2, \tag{26}$$

and therefore,

$$\operatorname{diag}(\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}^T\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]) = \alpha^2 \operatorname{diag}(\boldsymbol{P}_{tr}^2) + (\alpha - \alpha^2)\boldsymbol{C}_{tr}^2 = \alpha^2\boldsymbol{Q}_{tr}^2 + (\alpha - \alpha^2)\boldsymbol{C}_{tr}^2. \tag{27}$$

- For the off-diagonal entries, where $i \neq j$, we have

$$\left(\sum_{k} \mathop{\mathbb{E}}_{splits}[M_{ii}M_{jj}M_{kk}]P_{ki}P_{kj}\right)_{ij} = \left(\alpha^3 \sum_{k\neq i \wedge k \neq j} P_{tr,ki}P_{tr,kj} + \alpha^2(P_{tr,ii}P_{tr,ij} + P_{tr,ji}P_{tr,jj})\right)_{ij}. \tag{28}$$

So $\mathbb{E}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}^T\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]$ (ignore the diagonal entries) is

$$\alpha^3\boldsymbol{P}_{tr}^2 + (\alpha^2 - \alpha^3)(\boldsymbol{C}_{tr}\boldsymbol{P}_{tr} + \boldsymbol{P}_{tr}\boldsymbol{C}_{tr}). \tag{29}$$

From (27) and (29), we know that

$$\begin{aligned}&\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}^T\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\\ =&\alpha^3\boldsymbol{P}_{tr}^2 + (\alpha^2 - \alpha^3)\boldsymbol{Q}_{tr}^2 + (\alpha - 3\alpha^2 + 2\alpha^3)\boldsymbol{C}_{tr}^2 + (\alpha^2 - \alpha^3)(\boldsymbol{C}_{tr}\boldsymbol{P}_{tr} + \boldsymbol{P}_{tr}\boldsymbol{C}_{tr}).\end{aligned} \tag{30}$$

Hopefully, $\boldsymbol{C}_{tr}\boldsymbol{P}_{tr} + \boldsymbol{P}_{tr}\boldsymbol{C}_{tr}$ is positive semi-definite so that it can be decomposed. However, we found that this may not hold in some cases, especially when α is close to 1.
Then

$$\begin{aligned}&\operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T E[\boldsymbol{M}_{in}^T\boldsymbol{P}_{tr}\boldsymbol{M}_{in}\boldsymbol{P}_{tr}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}]\\ &= \operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T(\alpha^3\boldsymbol{P}_{tr}^2 + (\alpha^2 - \alpha^3)\operatorname{diag}(\boldsymbol{P}_{tr}^2) + (\alpha^2 - \alpha^3)(\boldsymbol{C}_{tr}\boldsymbol{P}_{tr} + \boldsymbol{P}_{tr}\boldsymbol{C}_{tr})\\ &\quad + (\alpha^2 - \alpha^3)\operatorname{diag}((\boldsymbol{C}_{tr}\boldsymbol{P}_{tr} + \boldsymbol{P}_{tr}\boldsymbol{C}_{tr})) + (\alpha - \alpha^2)\boldsymbol{C}_{tr}^2)\boldsymbol{Y}_{tr}\boldsymbol{W}]\\ &= \operatorname{tr}[\alpha^3\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\boldsymbol{P}_{tr}^2\boldsymbol{Y}_{tr}\boldsymbol{W} + (\alpha^2 - \alpha^3)\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\operatorname{diag}(\boldsymbol{P}_{tr}^2)\boldsymbol{Y}_{tr}\boldsymbol{W}\\ &\quad + (\alpha^2 - \alpha^3)\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T(\boldsymbol{C}_{tr}\boldsymbol{P}_{tr} + \boldsymbol{P}_{tr}\boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W} + (\alpha - 3\alpha^2 + 2\alpha^3)\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T\boldsymbol{C}_{tr}^2\boldsymbol{Y}_{tr}\boldsymbol{W}]\\ &= \alpha^3\|\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + (\alpha^2 - \alpha^3)\|\boldsymbol{Q}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + (\alpha - 3\alpha^2 + 2\alpha^3)\|\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2\\ &\quad + (\alpha^2 - \alpha^3)\operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T(\boldsymbol{C}_{tr}\boldsymbol{P}_{tr} + \boldsymbol{P}_{tr}\boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}].\end{aligned} \tag{31}$$

Combining the three terms, we get

$$\begin{aligned}&\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2]\\ &= (\alpha^2 - \alpha^3)\|\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + (\alpha - 2\alpha^2 + \alpha^3)\|\boldsymbol{Q}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2\\ &\quad - (\alpha^2 - \alpha^3)\operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}(\boldsymbol{P}_{tr}\boldsymbol{C}_{tr} + \boldsymbol{C}_{tr}\boldsymbol{P}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}] - (\alpha - 3\alpha^2 + 2\alpha^3)\|\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2.\end{aligned} \tag{32}$$

The overall result $\mathcal{L}(\boldsymbol{W})$ is

$$\begin{aligned}&\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}\|_F^2]\\ &= \mathop{\mathbb{E}}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{Y}_{tr}\|_F^2] - \frac{2}{\alpha}\mathop{\mathbb{E}}_{splits}[\operatorname{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}]] + \frac{1}{\alpha^2}\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{M}_{in}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2]\\ &= (1 - \alpha)\|\boldsymbol{Y}_{tr}\|_F^2 - 2(1 - \alpha)\operatorname{tr}[\boldsymbol{Y}_{tr}^T(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}] + (1 - \alpha)\|\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2\\ &\quad + (\frac{1}{\alpha} - 2 + \alpha)\|\boldsymbol{Q}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 - (1 - \alpha)\operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}(\boldsymbol{P}_{tr}\boldsymbol{C}_{tr} + \boldsymbol{C}_{tr}\boldsymbol{P}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}]\\ &\quad - (\frac{1}{\alpha} - 3 + 2\alpha)\|\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2,\end{aligned} \tag{33}$$

where $\boldsymbol{C}_{tr} = \operatorname{diag}(\boldsymbol{P}_{tr})$ and $\boldsymbol{Q}_{tr} = \operatorname{diag}(\boldsymbol{P}_{tr}^2)^{\frac{1}{2}}$.

Next we compute $\|\boldsymbol{Y}_{tr} - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2$.

Since

$$\begin{aligned}
&\|(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 \\
&= \operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}^T(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})^T(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}] \\
&= \|\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + \|\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 - \operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}(\boldsymbol{P}_{tr}\boldsymbol{C}_{tr} + \boldsymbol{C}_{tr}\boldsymbol{P}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}],
\end{aligned} \tag{34}$$

we have that

$$\begin{aligned}
&\|\boldsymbol{Y}_{tr} - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 \\
&= \|\boldsymbol{Y}_{tr}\|_F^2 + \|\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + \|\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 \\
&\quad - \operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}(\boldsymbol{P}_{tr}\boldsymbol{C}_{tr} + \boldsymbol{C}_{tr}\boldsymbol{P}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}] - 2\operatorname{tr}[\boldsymbol{Y}_{tr}^T(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}].
\end{aligned} \tag{35}$$

Therefore,

$$\begin{aligned}
&\frac{1}{1-\alpha}\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}\|_F^2] \\
&= \|\boldsymbol{Y}_{tr}\|_F^2 - 2\operatorname{tr}[\boldsymbol{Y}_{tr}^T(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}] + \|\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 \\
&\quad + \frac{1-\alpha}{\alpha}\|\boldsymbol{Q}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 - \operatorname{tr}[\boldsymbol{W}^T\boldsymbol{Y}_{tr}(\boldsymbol{P}_{tr}\boldsymbol{C}_{tr} + \boldsymbol{C}_{tr}\boldsymbol{P}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}] + \frac{2\alpha-1}{\alpha}\|\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 \\
&= \|\boldsymbol{Y}_{tr} - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + \frac{1-\alpha}{\alpha}\|\boldsymbol{Q}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 - \frac{1-\alpha}{\alpha}\|\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2.
\end{aligned} \tag{36}$$

Let $\boldsymbol{U}_{tr} = (\boldsymbol{Q}_{tr}^2 - \boldsymbol{C}_{tr})^{\frac{1}{2}} = (\operatorname{diag}(\boldsymbol{P}_{tr}^2) - \boldsymbol{C}_{tr}^2)^{\frac{1}{2}}$, and $\boldsymbol{\Gamma} = \boldsymbol{U}_{tr}\boldsymbol{Y}_{tr}$, then

$$\begin{aligned}
\frac{1}{1-\alpha}\mathop{\mathbb{E}}_{splits}[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}\|_F^2] &= \|\boldsymbol{Y}_{tr} - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + \frac{1-\alpha}{\alpha}\|\boldsymbol{U}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 \\
&= \|\boldsymbol{Y}_{tr} - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}\|_F^2 + \frac{1-\alpha}{\alpha}\|\boldsymbol{\Gamma}\boldsymbol{W}\|_F^2.
\end{aligned} \tag{37}$$

A.2 Proof of Corollary 1

We have

$$\begin{aligned}
&\frac{1}{1-\alpha}\mathop{\mathbb{E}}_{splits}\left[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}_y\|_F^2\right] \\
&= \frac{1}{1-\alpha}\mathop{\mathbb{E}}_{splits}\Big[\|\boldsymbol{Y}_{out} - \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}_y\|_F^2 \\
&\quad + \|\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x\|_F^2 - 2\operatorname{tr}[\boldsymbol{Y}_{out}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x] + \frac{2}{\alpha}\operatorname{tr}[\boldsymbol{W}_y^T\boldsymbol{Y}_{in}^T\boldsymbol{P}^T\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x]\Big] \\
&= \|\boldsymbol{Y}_{tr} - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y\|_F^2 + \frac{1-\alpha}{\alpha}\|\boldsymbol{\Gamma}\boldsymbol{W}_y\|_F^2 \\
&\quad + \|\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x\|_F^2 - 2\operatorname{tr}[\boldsymbol{Y}_{tr}^T\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x] + 2\operatorname{tr}[\boldsymbol{W}_y^T\boldsymbol{Y}_{tr}^T(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x] \\
&= \|\boldsymbol{Y}_{tr} - \boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x - (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y\|_F^2 + \frac{1-\alpha}{\alpha}\|\boldsymbol{\Gamma}\boldsymbol{W}_y\|_F^2.
\end{aligned} \tag{38}$$

A.3 Proof of Corollary 2

It is not hard to note that

$$
\begin{aligned}
\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}_y] &= \frac{1}{\alpha}\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{Y}_{in}\boldsymbol{W}_y] - \frac{1}{\alpha}\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{Y}_{in}\boldsymbol{W}_y] \\
&= \frac{1}{\alpha}\boldsymbol{M}_{tr}\boldsymbol{P}\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}_y - \frac{1}{\alpha}\mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{in}\boldsymbol{P}\boldsymbol{M}_{in}]\boldsymbol{Y}_{tr}\boldsymbol{W}_y \\
&= \boldsymbol{M}_{tr}\boldsymbol{P}\boldsymbol{Y}_{tr}\boldsymbol{W}_y - (\alpha\boldsymbol{P}_{tr} - \alpha\boldsymbol{C}_{tr} + \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y \\
&= (1-\alpha)\boldsymbol{M}_{tr}\boldsymbol{P}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}_y - (1-\alpha)\boldsymbol{C}_{tr}\boldsymbol{Y}_{tr}\boldsymbol{W}_y \\
&= (1-\alpha)\boldsymbol{M}_{tr}(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y \\
&= (1-\alpha)(\boldsymbol{M}_{tr}(\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y).
\end{aligned} \tag{39}
$$

Then by Jensen's inequality we have

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{splits}[\text{CrossEntropy}_{\mathcal{D}_{out}}(\boldsymbol{Y}_{out}, \boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x + \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}_y)] \\
&\geq \text{CrossEntropy}_{\mathcal{D}_{out}}(\boldsymbol{Y}_{out}, \mathop{\mathbb{E}}_{splits}[\boldsymbol{M}_{out}\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x + \boldsymbol{M}_{out}\boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}_y]) \\
&= \text{CrossEntropy}_{\mathcal{D}_{out}}(\boldsymbol{Y}_{tr}, (1-\alpha)(\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x + (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y))).
\end{aligned} \tag{40}
$$

Notice the nature of cross-entropy, if $0 < \alpha < 1$, we have

$$
\text{CrossEntropy}(\boldsymbol{Z}, (1-\alpha)\tilde{\boldsymbol{Z}}) \geq (1-\alpha)\text{CrossEntropy}(\boldsymbol{Z}, \tilde{\boldsymbol{Z}}). \tag{41}
$$

Therefore,

$$
\begin{aligned}
&\frac{1}{1-\alpha}\mathop{\mathbb{E}}_{splits}[\text{CrossEntropy}_{\mathcal{D}_{out}}(\boldsymbol{Y}_{out}, \boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_x + \boldsymbol{P}\tilde{\boldsymbol{Y}}_{in}\boldsymbol{W}_y)] \\
&\geq \text{CrossEntropy}_{\mathcal{D}_{tr}}(\boldsymbol{Y}_{tr}, \boldsymbol{X}\boldsymbol{W}_x + (\boldsymbol{P}_{tr} - \boldsymbol{C}_{tr})\boldsymbol{Y}_{tr}\boldsymbol{W}_y).
\end{aligned} \tag{42}
$$

B Experimental Details

B.1 Trainable Label Propagation

In Table 1, we employ trainable label propagation with the same settings as the original label propagation algorithm except the label trick, such as the trade-off term λ in (1) and the number of propagation steps. We search the best splitting probability α in range of $\{0.05, 0.1, \ldots, 0.95\}$. Note that for the first two datasets, we use a random splitting strategy to divide the whole dataset into training, validation, and test datasets; while for the last two datasets, we use the standard split. For each dataset, we run 10 runs with different random seeds and report the results of mean and variance.

B.2 Deterministic Label Trick applied to GNNs with Linear Layers

As it is merely a comparison of the two methods, in Table 3, we do not spend effort to tune the model to achieve the best results. We simply choose one specific set of hyperparameters and run the model on each dataset with or without label trick, except for TWIRLS for which we used the best hyperparameters. Also, for SIGN we used a simplified version for speeding up the training. Specifically, for TWIRLS, we use the hyperparameter in the original from (Yang et al., 2021). For SGC, the number of propagation steps is set to 3, and there is one extra linear transformation after the propagation steps. For SIGN, we use 5 propagation steps, and tune the number of MLP layers from 1 or 2 on each dataset. Also for SIGN, we sum up all immediate results instead of concatenating them to simplify the implementation. We uniformly use 2 propagation steps for all linear propagation layers, and for TWIRLS, λ is set to 1.

B.3 Effect of the Splitting Probability

In Figure 1, we use a linear model similar to the trainable label propagation algorithm with features and labels as input and a 3-layer GCN. Each model uses the same hyperparameters except for the

splitting probability α, and the number of epochs, because when α is close to zero or one, the model needs more epochs to converge. For the linear model, λ is set to 0.9 and the number of propagation steps is 9.

B.4 Trainable Correct and Smooth

We begin by restating the predictor formulation of trainable Correct and Smooth in (12):

$$\begin{aligned} f_{TC\&S}(\tilde{\boldsymbol{Y}};\boldsymbol{\mathcal{W}}) &= \boldsymbol{P}_s(\boldsymbol{Y}_{in} + (\boldsymbol{M}_{te} + \boldsymbol{M}_{out})(\tilde{\boldsymbol{Y}} + \tilde{\boldsymbol{E}}_{in}\boldsymbol{W}_c))\boldsymbol{W}_s \\ &= \boldsymbol{P}_s((\boldsymbol{Y}_{in} + (\boldsymbol{M}_{te} + \boldsymbol{M}_{out})\tilde{\boldsymbol{Y}})\boldsymbol{W}_s + \boldsymbol{P}_s(\boldsymbol{M}_{te} + \boldsymbol{M}_{out})\tilde{\boldsymbol{E}}_{in}\boldsymbol{W}_c\boldsymbol{W}_s \\ &= \underbrace{\boldsymbol{P}_s((\boldsymbol{Y}_{in} + (\boldsymbol{M}_{te} + \boldsymbol{M}_{out})\tilde{\boldsymbol{Y}})}_{\hat{\boldsymbol{Y}}_s}\boldsymbol{W}_s + \underbrace{\boldsymbol{P}_s(\boldsymbol{M}_{te} + \boldsymbol{M}_{out})\tilde{\boldsymbol{E}}_{in}}_{\hat{\boldsymbol{Y}}_c}\hat{\boldsymbol{W}}_c, \end{aligned} \tag{43}$$

Note that the computation of $\tilde{\boldsymbol{Y}}_s$ and $\tilde{\boldsymbol{Y}}_c$ does not involve any trainable parameters. Therefore, we can compute them beforehand for each training split.

We now describe our experimental setup in Table 4. We first trained an MLP with exactly the same hyperparameters as in Huang et al. (2020). For each splitting probability $\alpha \in \{0.1, 0.2, \cdots, 0.9\}$, we generate 10 splits and precompute $\tilde{\boldsymbol{Y}}_s$ and $\tilde{\boldsymbol{Y}}_c$. Then for each training epoch, we cycle over the set of $\tilde{\boldsymbol{Y}}_s$ and $\tilde{\boldsymbol{Y}}_c$. We use cross-entropy as the loss function in (13), and trained the weight matrices using Adam optimizer with a learning rate of 0.1. We used 20 propagation steps for both Correct and Smooth steps.