Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification

Yunsheng Shi, **Zhengjie Huang**, **Shikun Feng**, **Hui Zhong**, **Wenjing Wang**, **Yu Sun**
Baidu Inc., China
{shiyunsheng01, huangzhengjie, fengshikun01, zhonghui03, wangwenjin02, sunyu02}@baidu.com

Abstract

Graph neural network (GNN) and label propagation algorithm (LPA) are both message passing algorithms, which have achieved superior performance in semi-supervised classification. GNN performs *feature propagation* by a neural network to make predictions, while LPA uses *label propagation* across graph adjacency matrix to get results. However, there is still no effective way to directly combine these two kinds of algorithms. To address this issue, we propose a novel **Uni**fied **M**essage **P**assaging Model (UniMP) that can incorporate *feature* and *label propagation* at both training and inference time. First, UniMP adopts a Graph Transformer network, taking feature embedding and label embedding as input information for propagation. Second, to train the network without overfitting in self-loop input label information, UniMP introduces a masked label prediction strategy, in which some percentage of input label information are masked at random, and then predicted. UniMP conceptually unifies feature propagation and label propagation and is empirically powerful. It obtains new state-of-the-art semi-supervised classification results in Open Graph Benchmark (OGB).

1 Introduction

There are various scenarios in the world, e.g., recommending related news, discovering new drugs, or predicting social relations, which can be described as graph structures. And many methods have been proposed to optimize these graph-based problems and achieved significant success in many related domains such as predicting nodes' properties [Yang *et al.*, 2016; Kipf and Welling, 2016], relation linking [Grover and Leskovec, 2016; Battaglia *et al.*, 2018], and graph classification [Duvenaud *et al.*, 2015; Niepert *et al.*, 2016].

In the task of semi-supervised node classification, we are required to learn with labeled examples and then make predictions for those unlabeled ones. To better classify the nodes' labels in the graph, based on the Laplacian smoothing assumption [Li *et al.*, 2018; Xu *et al.*, 2018b], the message passing models were proposed to aggregate the information from its connected neighbors in the graph, acquiring enough facts to produce a more robust prediction for unlabeled nodes. Generally, there are two main kinds of methods to implement message passing model, Graph Neural Networks (GNNs) [Kipf and Welling, 2016; Hamilton *et al.*, 2017; Xu *et al.*, 2018b; Liao *et al.*, 2019; Xu *et al.*, 2018a] and Label Propagation Algorithms (LPAs) [Zhu *et al.*, 2003; Zhang and Lee, 2007; Wang and Zhang, 2007; Karasuyama and Mamitsuka, 2013; Gong *et al.*, 2016; Liu *et al.*, 2019]. GNNs combine graph structures by propagating and aggregating node features through several neural layers, getting predictions from *feature propagation*. While LPAs make predictions for unlabeled instances by *label propagation* iteratively.

Model	Training		Inference	
	Feature	**Label**	**Feature**	**Label**
LPA		✓		✓
GCN	✓		✓	
APPNP	✓		✓	
GCN-LPA	✓	✓	✓	
UniMP (Ours)	✓	✓	✓	✓

Table 1: Comparison the input information that message passing models use in training and inference.

Since GNN and LPA are based on the same assumption, making semi-supervised classifications by information propagation, there is an intuition that incorporating them together for boosting performance. Some superior studies have proposed their graph models based on it. For example, APPNP [Klicpera *et al.*, 2018] and TPN [Liu *et al.*, 2019] using GNN predict soft labels and then propagate them, and GCN-LPA [Wang and Leskovec, 2019] uses LPA to regularize their GNN model. However, as shown in Table 1, aforementioned methods still can not directly incorporate GNN and LPA within a message passing model, *propagating feature* and *label* in both training and inference procedure.

In this work, we propose a **Uni**fied **M**essage **P**assing model (UniMP) to address the aforementioned issue with two simple but effective ideas: (a) combing node features propagation with labels and (b) masked label prediction. Previous GNN-based methods only take node features as input with the partial observed node labels for supervised training. And they discard the observed labels during inference. UniMP utilizes both node features and labels in both training and inference stages. It uses the embedding technique to transform the partial node labels from one-hot to dense vector likes node fea-

tures. And a multi-layer Graph Transformer network takes them as input to perform attentive information propagation between nodes. Therefore, each node can aggregate both features and labels information from its neighbors. Since we have taken the node label as input, using it for supervised training will cause the label leakage problem. The model will overfit in the self-loop input label while performing poor in inference. To address this issue, we propose a masked label prediction strategy, which randomly masks some training instances' label and then predicts them to overcome label leakage. This simple and effective training method is drawn the lesson from masked word prediction in BERT [Devlin *et al.*, 2018], and simulates the procedure of transducing labels information from labeled to unlabeled examples in the graph.

We evaluate our UniMP model on three semi-supervised classification datasets in the Open Graph Benchmark (OGB), where our new methods achieve the new state-of-the-art results in all tasks, gaining 82.56% ACC in *ogbn-products*, 86.42% ROC-AUC in *ogbn-proteins* and 73.11% ACC in *ogbn-arxiv*. We also conduct ablation studies for our UniMP model, to evaluate the effectiveness of our unified method. Besides, we make the most thorough analysis of how the label propagation boosts our model's performance.

2 Preliminaries

In this section, we briefly review the related work and along the way, introduce our notation. We denote a graph as $G = (V, E)$, where V denotes the nodes in the graph with $|V| = n$ and E denotes edges. The nodes are described by the feature matrix $X \in \mathbb{R}^{n \times m}$, which usually are dense vectors with m dimension, and the target class matrix $Y \in \mathbb{R}^{n \times c}$, with the number of classes c. The adjacency matrix $A = [a_{i,j}] \in \mathbb{R}^{n \times n}$ is used to describe graph G, and the diagonal degree matrix is denoted by $D = \text{diag}(d_1, d_2, ..., d_n)$, where $d_i = \sum_j a_{ij}$ is the degree of node i. A normalized adjacency matrix is defined as $D^{-1}A$ or $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, and we adopt the first definition in this paper.

Graph Neural Networks. In semi-supervised node classification, GCN [Kipf and Welling, 2016] is one of the most classical models based on the Laplacian smoothing assumption. GCN transforms and propagates node features X across the graph by several layers, including linear layers and nonlinear activation to build the approximation of the mapping: $X \to Y$. The feature propagation scheme of GCN in layer l is:

$$\begin{aligned} H^{(l+1)} &= \sigma(D^{-1}AH^{(l)}W^{(l)}) \\ Y &= f_{out}(H^{(L)}) \end{aligned} \tag{1}$$

where the σ is an activation function, $W^{(l)}$ is the trainable weight in the l-th layer, and the $H^{(l)}$ is the l-th layer representations of nodes. $H^{(0)}$ is equal to node input features X. Finally, a f_{out} output layer is applied on the final representation to make prediction for Y.

Label propagation algorithms. Traditional algorithms like Label Propagation Algorithm (LPA) only utilizes labels and relations between nodes to make prediction. LPA assumes the labels between connected nodes are similar and propagates the labels iteratively across the graph. Given an initial label matrix $\hat{Y}^{(0)}$, which consists of one-hot label indicator vectors $\hat{y}_i^0$ for the labeled nodes or zeros vectors for the unlabeled. A simple iteration equation of LPA is formulated as following:

$$\hat{Y}^{(l+1)} = D^{-1}A\hat{Y}^{(l)} \tag{2}$$

Labels are propagated from each other nodes through a normalized adjacency matrix $D^{-1}A$.

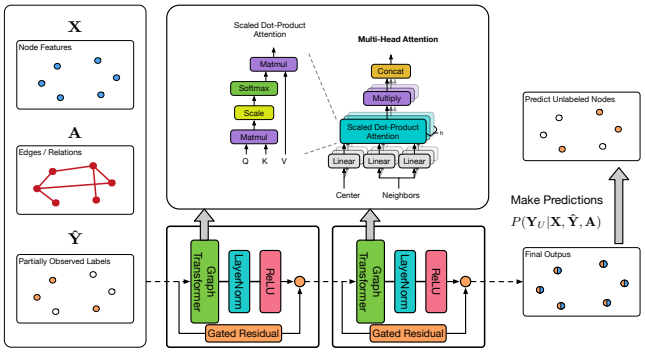

Figure 1: The architecture of UniMP.

Combining GNN and LPA. Recently, there is a trend to combine GNN and LPA in semi-classification tasks in the community. APPNP [Klicpera *et al.*, 2018] and TPN [Liu *et al.*, 2019] propose to use GCN to predict soft labels and then propagate them with Personalized Pagerank. However, these works still only considered the partial node labels as the supervision training signal. GCN-LPA is most relevant to our work, as they also take the partial node labels as input. However, they combine the GNN and LPA in a more indirect way, only using the LPA in training to regularize the weight edges of their GAT model. While our UniMP directly combines GNN and LPA within a network, propagates the node features and labels in both training and predicting. Moreover, unlike GCN-LPA whose regularization strategy can only be used in those GNNs with trainable weight edge such as GAT [Veličković *et al.*, 2017], GAAN [Zhang *et al.*, 2018], our training strategy can be easily extended in kinds of GNNs such as GCN and GAT to further improve their performance. We will describe our approach more specifically in the next section.

3 Unified Message Passing Model

As shown in Figure 1, given the node feature X and partial observed labels $\hat{Y}$, we employ a Graph Transformer, jointly using label embedding to combine the aforementioned feature and label propagation together, constructing our UniMP model. Moreover, a masked label prediction strategy is introduced to train our model to prevent label leakage problem.

3.1 Graph Transformer

Since Transformer [Vaswani *et al.*, 2017; Devlin *et al.*, 2018] has been proved being powerful in NLP, we adopt its vanilla

multi-head attention into graph learning with taking into account the case of edge features. Specifically, given node features $H^{(l)} = \{h_1^{(l)}, h_2^{(l)}, ..., h_n^{(l)}\}$, we calculate multi-head attention for each edge from j to i as following:

$$\begin{aligned} q_{c,i}^{(l)} &= W_{c,q}^{(l)} h_i^{(l)} + b_{c,q}^{(l)} \\ k_{c,j}^{(l)} &= W_{c,k}^{(l)} h_j^{(l)} + b_{c,k}^{(l)} \\ e_{c,ij} &= W_{c,e} e_{ij} + b_{c,e} \\ \alpha_{c,ij}^{(l)} &= \frac{\langle q_{c,i}^{(l)}, k_{c,j}^{(l)} + e_{c,ij} \rangle}{\sum_{u \in \mathcal{N}(i)} \langle q_{c,i}^{(l)}, k_{c,u}^{(l)} + e_{c,iu} \rangle} \end{aligned} \tag{3}$$

where $\langle q, k \rangle = \exp(\frac{q^T k}{\sqrt{d}})$ is exponential scale dot-product function and d is the hidden size of each head. For the c-th head attention, we firstly transform the source feature $h_i^{(l)}$ and distant feature $h_j^{(l)}$ into query vector $q_{c,i}^{(l)} \in \mathbb{R}^d$and key vector $k_{c,j}^{(l)} \in \mathbb{R}^d$ respectively using different trainable parameters $W_{c,q}^{(l)}$, $W_{c,k}^{(l)}$, $b_{c,q}^{(l)}$, $b_{c,k}^{(l)}$. The provided edge features e_{ij} will be encoded and added into key vector as additional information for each layer.

After getting the graph multi-head attention, we make a message aggregation from the distant j to the source i:

$$\begin{aligned} v_{c,j}^{(l)} &= W_{c,v}^{(l)} h_j^{(l)} + b_{c,v}^{(l)} \\ \hat{h}_i^{(l+1)} &= \Big\|_{c=1}^{C} \big[\sum_{j \in \mathcal{N}(i)} \alpha_{c,ij}^{(l)} (v_{c,j}^{(l)} + e_{c,ij}) \big] \end{aligned} \tag{4}$$

where the $\|$ is the concatenation operation for C head attention. Comparing with the Equation 1, multi-head attention matrix replaces the original normalized adjacency matrix as transition matrix for message passing. The distant feature h_j is transformed to $v_{c,j} \in \mathbb{R}^d$ for weighted sum.

In addition, inspired by Li [2019] and Chen [2020], we propose to use a gated residual connection between layers as shown in Equation 5 to prevent our model from over-smoothing.

$$\begin{aligned} r_i^{(l)} &= W_r^{(l)} h_i^{(l)} + b_r^{(l)} \\ \beta_i^{(l)} &= \text{sigmoid}(W_g^{(l)} [\hat{h}_i^{(l+1)}; r_i^{(l)}; \hat{h}_i^{(l+1)} - r_i^{(l)}]) \\ h_i^{(l+1)} &= \text{ReLU}(\text{LayerNorm}((1 - \beta_i^{(l)}) \hat{h}_i^{(l+1)} + \beta_i^{(l)} r_i^{(l)})) \end{aligned} \tag{5}$$

Specially, similar to GAT, if we apply the Graph Transformer on the last output layer, we will employ averaging for multi-head output and remove the non-linear transformation as following:

$$\begin{aligned} \hat{h}_i^{(l+1)} &= \frac{1}{C} \sum_{c=1}^{C} \big[\sum_{j \in \mathcal{N}(i)} \alpha_{c,ij}^{(l)} (v_{c,j}^{(l)} + e_{c,ij}^{(l)}) \big] \\ h_i^{(l+1)} &= (1 - \beta_i^{(l)}) \hat{h}_i^{(l+1)} + \beta_i^{(l)} r_i^{(l)} \end{aligned} \tag{6}$$

3.2 Label Embedding and Propagation

We propose to embed the partially observed labels into the same space as node features: $\hat{Y} \in \mathbb{R}^{n \times c} \rightarrow \hat{Y}_d \in \mathbb{R}^{n \times m}$, which consist of the label embedding vector for labeled nodes and zeros vectors for the unlabeled. And then, we combine the label propagation into Graph Transformer by simply adding the node features and labels vectors together as propagation information $(H^0 = X + \hat{Y}_d) \in \mathbb{R}^{n \times m}$. We can prove that by mapping partially-labeled $\hat{Y}$ and node features X into the same space and adding them up, our model is unifying both label propagation and feature propagation within a shared message passing framework. Let's take $\hat{Y}_d = \hat{Y} W_d$ and A^* to be normalized adjacency matrix $D^{-1}A$ or the attention matrix from our Graph Transformer likes Equation 3. Then we can find that:

$$\begin{aligned} H^{(0)} &= X + \hat{Y} W_d \\ H^{(l+1)} &= \sigma(((1 - \beta) A^* + \beta I) H^{(l)} W^{(l)}) \end{aligned} \tag{7}$$

where β can be the gated function like Equation 5 or a predefined hyper-parameters like APPNP [Klicpera *et al.*, 2018]. For simplification, we let σ function as identity function, then we can get:

$$\begin{aligned} H^{(l)} =& ((1 - \beta) A^* + \beta I)^l (X + \hat{Y} W_d) W^{(1)} W^{(2)} \dots W^{(l)} \\ =& ((1 - \beta) A^* + \beta I)^l X W + ((1 - \beta) A^* + \beta I)^l \hat{Y} W_d W \end{aligned} \tag{8}$$

where $W = W^{(1)} W^{(2)} \dots W^{(l)}$. Then we can find that our model can be approximately decomposed into feature propagation $((1 - \beta) A^* + \beta I)^l X W$ and label propagation $((1 - \beta) A^* + \beta I)^l \hat{Y} W_d W$.

3.3 Masked Label Prediction

Previous works on GNNs seldom consider using the partially observed labels $\hat{Y}$ in both training and inference stages. They only take those labels information as ground truth target to supervised train their model's parameters θ with given X and A:

$$\arg\max_{\theta} \quad \log p_\theta(\hat{Y} | X, A) = \sum_{i=1}^{\hat{V}} \log p_\theta(\hat{y}_i | X, A) \tag{9}$$

where $\hat{V}$ represents the partial nodes with labels. However, our UniMP model propagates node features and labels to make prediction: $p(y | X, \hat{Y}, A)$. Simply using above objective for our model will make the label leakage in the training stage, causing poor performance in inference. Learning from BERT, which masks input words and makes predictions for them to pretrain their model (masked word prediction), we propose a masked label prediction strategy to train our model. During training, at each step, we corrupt the $\hat{Y}$ into $\tilde{Y}$ by randomly masking a portion of node labels to zeros and keep the others remain, which is controlled by a hyper-parameter called label_rate. Let those masked labels be $\bar{Y}$, our objective function is to predict $\bar{Y}$ with given X, $\tilde{Y}$ and A:

$$\arg\max_{\theta} \quad \log p_\theta(\bar{Y} | X, \tilde{Y}, A) = \sum_{i=1}^{\bar{V}} \log p_\theta(\bar{y}_i | X, \tilde{Y}, A) \tag{10}$$

where $\bar{V}$ represents those nodes with masked labels. In this way, we can train our model without the leakage of self-loop labels information. And during inference, we will employ all $\hat{Y}$ as input labels to predict the remaining unlabeled nodes.

4 Experiments

We propose a Unified Message Passing Model (UniMP) for semi-supervised node classification, which incorporates the feature and label propagation jointly by a Graph Transformer and employs a masked label prediction strategy to optimize it. We conduct the experiments on the Node Property Prediction of Open Graph Benchmark (OGBN), which includes several various challenging and large-scale datasets for semi-supervised classification, split in the procedure that closely matches the real-world application [Hu *et al.*, 2020]. To verify our models effectiveness, we compare our model with others state-of-the-art (SOTA) models in *ogbn-products*, *ogbn-proteins* and *ogbn-arxiv* three OGBN datasets. We also provide more experiments and comprehensive ablation studies to show our motivation more intuitively, and how LPA improves our model to achieve better results.

4.1 Datasets and Experimental Settings

Name	Node	Edges	Tasks	Task Type	Metric
ogbn-products	2,449,029	61,859,140	1	Multi-class class	Accuracy
ogbn-proteins	132,534	39,561,252	112	Binary class	ROC-AUC
ogbn-arxiv	169,343	1,166,243	1	Multi-class class	Accuracy

Table 2: Dataset statistics of OGB node property prediction

Datasets. Most of the frequently-used graph datasets are extremely small compared to graphs found in real applications. And the performance of GNNs on these datasets is often unstable due to several issues including their small-scale nature, non-negligible duplication or leakage rates, unrealistic data splits [Hu *et al.*, 2020]. Consequently, we conduct our experiments on the recently released datasets of Open Graph Benchmark (OGB) [Hu *et al.*, 2020], which overcome the main drawbacks of commonly used datasets and thus are much more realistic and challenging. OGB datasets cover a variety of real-world applications and span several important domains ranging from social and information networks to biological networks, molecular graphs, and knowledge graphs. They also span a variety of prediction tasks at the level of nodes, graphs, and links/edges. As shown in table 2, in this work, we performed our experiments on the three OGBN datasets with different sizes and tasks for getting credible result, including *ogbn-products* about 47 products categories classification with given 100-dimensional nodes features, *ogbn-proteins* about 112 kinds of proteins function prediction with given 8-dimensional edges features and *ogbn-arxiv* about 40-class topics classification with given 128 dimension nodes features. More details about these datasets are provided in appendix A in the supplementary file.

	ogbn-products	ogbn-proteins	ogbn-arxiv
sampling_method	NeighborSampling	Random Partition	Full-batch
num_layers	3	7	3
hidden_size	128	64	128
num_heads	4	4	2
dropout	0.3	0.1	0.3
lr	0.001	0.001	0.001
weight_decay	*	*	0.0005
label_rate	0.625	0.5	0.625

Table 3: The hyper-paramerter setting of our model

Implementation details. As mentioned above, these datasets are different from each other in sizes or tasks. So we evaluate our model on them with different sampling methods following previous studies [Li *et al.*, 2020], getting credible comparison results. In *ogbn-products* dataset, we use NeighborSampling with size =10 for each layer to sample the subgraph during training and use full-batch for inference. In *ogbn-proteins* dataset, we use Random Partition to split the dense graph into subgraph to train and test our model. As for small-size *ogbn-arxiv* dataset, we just apply full batch for both training and test. We set the hyper-parameter of our model for each dataset in Table 3, and the label_rate means the percentage of labels we preserve during applying masked label prediction strategy. We use Adam optimizer with lr = 0.001 to train our model. Specially, we set weight decay to 0.0005 for our model in small-size *ogbn-arxiv* dataset to prevent overfitting. More details about the tuned hyper-parameters are provided in appendix B in the supplementary file.

4.2 Comparison with SOTA Models

Baseline and other comparative SOTA models are provided by OGB leaderboard. And all these results are guaranteed to be reproducible with open source codes. Following the requirement of OGB, we run our experimental results for each dataset 10 times and report the mean and standard deviation. As shown in Table 4, Table 5, and Table 6, our unified model outperform all other comparative models in three OGBN datasets. Since most of the compared models only consider optimizing their models for the features propagation, these results demonstrate that incorporating label propagation into GNN models can bring significant improvements. Specifically, we gain 82.56% ACC in *ogbn-products*, 86.42% ROC-AUC in *ogbn-proteins*, which achieves about 0.6-1.6% absolute improvements compared to the newly SOTA methods like DeeperGCN [Li *et al.*, 2020]. In *ogbn-arxiv*, our method gains 73.11% ACC, achieve 0.37% absolute improvements compared to GCNII [Chen *et al.*, 2020], whose parameters are four times larger than ours.

Model	Test Accuracy	Validation Accuracy	Params
GCN-Cluster [Chiang *et al.*, 2019]	0.7897 ± 0.0036	0.9212 ± 0.0009	206,895
GAT-Cluster	0.7923 ± 0.0078	0.8985 ± 0.0022	1,540,848
GAT-NeighborSampling	0.7945 ± 0.0059	-	1,751,574
GraphSAINT [Zeng *et al.*, 2019]	0.8027 ± 0.0026	-	331,661
DeeperGCN [Li *et al.*, 2020]	0.8090 ± 0.0020	0.9238 ± 0.0009	253,743
UniMP	$\mathbf{0.8256 \pm 0.0031}$	$\mathbf{0.9308 \pm 0.0017}$	1,475,605

Table 4: Results for ogbn-products

Model	Test ROC-AUC	Validation ROC-AUC	Params
GaAN [Zhang *et al.*, 2018]	0.7803 ± 0.0073	-	-
GeniePath-BS [Liu *et al.*, 2020b]	0.7825 ± 0.0035	-	316,754
MWE-DGCN	0.8436 ± 0.0065	0.8973 ± 0.0057	538,544
DeepGCN [Li *et al.*, 2019]	0.8496 ± 0.0028	0.8921 ± 0.0011	2,374,456
DeeperGCN [Li *et al.*, 2020]	0.8580 ± 0.0017	0.9106 ± 0.0016	2,374,568
UniMP	$\mathbf{0.8642 \pm 0.0008}$	$\mathbf{0.9175 \pm 0.0007}$	1,909,104

Table 5: Results for ogbn-proteins

Model	Test Accuracy	Validation Accuracy	Param
DeeperGCN [Li *et al.*, 2020]	0.7192 ± 0.0016	0.7262 ± 0.0014	1,471,506
GaAN [Zhang *et al.*, 2018]	0.7197 ± 0.0024	-	1,471,506
DAGNN [Liu *et al.*, 2020a]	0.7209 ± 0.0025	-	1,751,574
JKNet [Xu *et al.*, 2018b]	0.7219 ± 0.0021	0.7335 ± 0.0007	331,661
GCNII [Chen *et al.*, 2020]	0.7274 ± 0.0016	-	2,148,648
UniMP	$\mathbf{0.7311 \pm 0.0021}$	$\mathbf{0.7450 \pm 0.0005}$	473,489

Table 6: Results for ogbn-arxiv

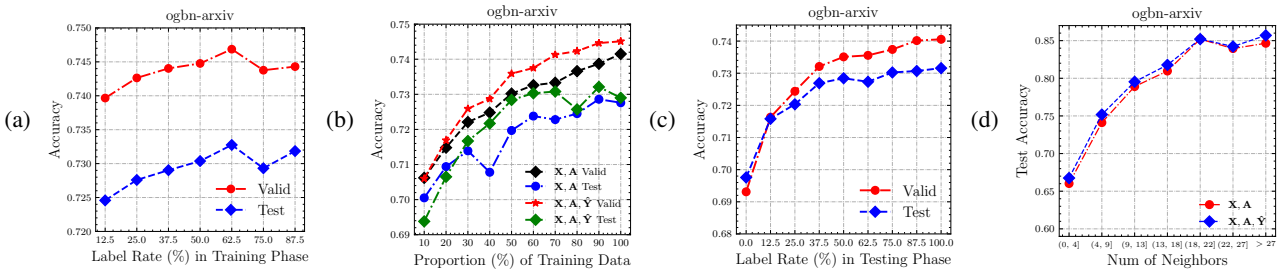

Figure 2: Exploration of how label coverage affects label propagation: (a) Training with different label_rate; (b) Training with different proportion of labeled data; (c) Testing with different label_rate; (d) Test accuracy with different neighbors.

Inputs	Model	Datasets		
		ogbn-products Test ACC	ogbn-proteins Test ROC-AUC	ogbn-arxiv Test ACC
$\mathbf{X}$	Multilayer Perceptron	0.6106 ± 0.0008	0.7204 ± 0.0048	0.5765 ± 0.0012
$\mathbf{X}, \mathbf{A}$	GCN	0.7851 ± 0.0011	0.8265 ± 0.0008	0.7218 ± 0.0014
	GAT	0.8002 ± 0.0063	0.8376 ± 0.0007	0.7246 ± 0.0013
	Graph Transformer	0.8137 ± 0.0047	0.8347 ± 0.0014	0.7292 ± 0.0010
$\mathbf{A}, \hat{\mathbf{Y}}$	GCN	0.7832 ± 0.0013	0.8083 ± 0.0021	0.7018 ± 0.0009
	GAT	0.7751 ± 0.0054	0.8247 ± 0.0033	0.7055 ± 0.0012
	Graph Transformer	0.7987 ± 0.0104	0.8160 ± 0.0007	0.7090 ± 0.0007
$\mathbf{X}, \mathbf{A}, \hat{\mathbf{Y}}$	GCN	0.7987 ± 0.0104	0.8247 ± 0.0032	0.7264 ± 0.0003
	GAT	0.8193 ± 0.0017	0.8556 ± 0.0009	0.7278 ± 0.0009
	Graph Transformer	$\mathbf{0.8256 \pm 0.0031}$	0.8560 ± 0.0003	$\mathbf{0.7311 \pm 0.0021}$
	⌞ w/ Edge Feature	*	$\mathbf{0.8642 \pm 0.0008}$	*

Table 7: This is the ablation studies on models with different inputs, where $\mathbf{X}$ denotes the nodes features, $\mathbf{A}$ is the graph adjacent matrix and $\hat{\mathbf{Y}}$ is the observed labels. In *ogbn-proteins*, nodes features are not provided initially. We average the edge features as their nodes features and provide the experimental result of Transformer without edge features for fair comparison in this experiment, which is slightly different from Table 5.

4.3 Ablation Studies

In this section, to better identify the improvements from different components of our proposed model, we conduct extensive studies with the following four aspects:

- Firstly, we apply the masked label prediction strategy on kinds of GNNS to show the effectiveness and robustness of incorporation LPA and GNN, shown in Table 7.
- In order to get a more practical and effective solution to apply masked label prediction strategy, we tune the label_rate during training and inference to explore the relationship between label coverage and GNNs performance, shown in Figure 2.
- We also analyze how LPA affects the GNN to make it performs better, shown in Figure 3.
- Furthermore, in Table 8, we provide more ablation studies on UniMP, compared with GAT, showing the superiority of our model.

Graph Neural Networks with Different Inputs

In Table 7, we apply masked label prediction on kinds of GNNs to improve their performance. Firstly, we re-implement classical GNN methods like GCN and GAT, following the same sampling methods and model setting shown in Table 3. The hidden size of GCN is head_num*hidden_size since it doesn't have head attention. Secondly, we change different inputs for these models to study the effectiveness of feature and label propagation, using our **masked label prediction** to train the models with partial nodes label $\hat{\mathbf{Y}}$ as input.

Row 4 in Table 7 shows that only with $\hat{\mathbf{Y}}$ and $\mathbf{A}$ as input, GNNs still work well in all three datasets, outperforming those MLP model only given $\mathbf{X}$. This implies that one's label relies heavily on its neighborhood instead of its feature. Comparing Row 3 and 5 in Table 7, models with $\mathbf{X}$, $\mathbf{A}$ and $\hat{\mathbf{Y}}$ outperform the models with $\mathbf{X}$ and $\mathbf{A}$, which indicates that it's a waste of information for GNNs in semi-supervised classification when they making predictions without incorporating the ground truth train labels $\hat{\mathbf{Y}}$. Row 3-5 in Table 7 also show that our Graph Transformer can outperform GAT, GCN with different input settings.

Relation between Label Coverage and Performance

Although we have verified the effectiveness of using this strategy to combine LPA and GNN, the relation between label coverage and its impact on GNNs performance remains uncertain. Therefore, shown in Figure 2, we conduct more experiments in *ogbn-arxiv* to investigate their relationship in the following different scenarios:

- In Figure 2a, we train UniMP using $\mathbf{X}, \hat{\mathbf{Y}}, \mathbf{A}$ as inputs. We tune the input label_rate which is the hyper-parameter of masked label prediction task and display the validation and test accuracy. Our model achieves better performance when label_rate is about 0.625.

- Figure 2b describes the correlation between the proportion of training data and the effectiveness of label propagation. We fix the input label_rate with 0.625. The only change is the training data proportion. It's common sense that with the increased amount of training data, the performance is gradually improving. And the model with label propagation $\hat{\mathbf{Y}}$ can gain greater benefits from increasing labeled data proportion.
- Our unified model always masks a part of the training input label and tries to recover them. But in the inference stage, our model utilizes all training labels for predictions, which is slightly inconsistent with the one in training. In Figure 2c, we fix our input label_rate with 0.625 during training and perform different input label_rate in inference. the training stage, It's found that UniMP might have worse performance (less than 0.70) than the baseline (about 0.72) when lowering the label_rate during prediction. However, when the label_rate climbs up, the performance can boost up to 0.73.
- In Figure 2d, we calculate the accuracy for unlabeled nodes grouped by the number of neighbors. The experimental result shows that nodes with more neighbors have higher accuracy. And the model with label propagation $\hat{\mathbf{Y}}$ can always have improvements even with different numbers of training neighbors.

Measuring the Connection between Nodes

In Figure 3, we analyze how LPA affects GNN to make it perform better. Wang [2019] has pointed out that using LPA for GCN during training can enable nodes within the same class/label to connect more strongly, increasing the accuracy (ACC) of model's prediction. Our model can be regarded as an upgraded version of them, using LPA in both training and testing time for our Graph Transformer. Therefore, we try to experimentally verify the above idea based on our model.

$$MSF = \frac{1}{N}\sum_{i=1}^{N}\log\left(1+\sum_{j\in\mathcal{N}(i)_{pos}}\sum_{k\in\mathcal{N}(i)_{neg}} e^{\alpha_{i,j}-e^{\alpha_{i,k}}}\right) \quad (11)$$

We use the Margin Similarity Function (MSF) as shown in Equation 11 to reflect the connection tightness between nodes within the same class (the higher scores, the stronger connection they have. We conduct the experiment on *ogbn-arxiv*. And as shown in Figure 3, the ACC of models' prediction is proportional to Margin Similarity. Unifying feature and label propagation can further strengthen their connection, improving their ACC. Moreover, our Graph Transformer outperforms GAT in both connection tightness and ACC with different inputs.

More Ablation Studies on UniMP

Finally, we provide more ablation studies on our UniMP model, compared with GAT, from the following 4 aspects: (1) vanilla transformer with dot-product attention or GAT with sum attention; (2) simple residual or gated residual; (3) with train labels as inputs; (4) with train and validation labels as inputs. As shown in Table 8, we can find that dot-product attention can outperform sum attention, since dot-product provides more interactions between nodes. Besides, residual and gated residual can also strengthen the GNNs with shallow layers. Moreover, our unified model can take the additional validation labels as input to further boost model's performance without more training steps. Therefore, when we apply the model to the real scene, and the labeled data are accumulated progressively, the accuracy of the unlabeled data can keep increasing without training our model from scratch, while other GNNs without explicit label modeling can't fully utilize the benefits of additional labels.

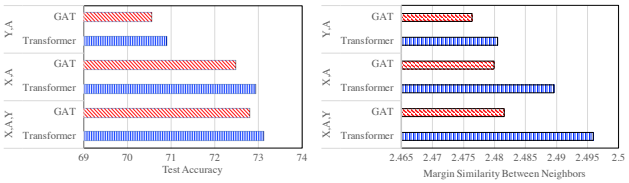

Figure 3: Correlation between accuracy and margin similarity between neighbors.

Model	**ogbn-prdouct**	**ogbn-arxiv**
GAT (sum attention)	0.8002	0.7246
⌞ w/ residual	0.8033	0.7265
⌞ w/ gated residual	0.8050	0.7272
Transformer (dot-product)	0.8091	0.7259
⌞ w/ residual	0.8125	0.7271
⌞ w/ gated residual	0.8137	0.7292
⌞ w/ train label (UniMP)	0.8256	0.7311
⌞ w/ validation labels	**0.8312**	**0.7377**

Table 8: Ablation studies in UniMP, compared with GAT

5 Conclusion

We first propose a unified message passing model, UniMP, which jointly performs feature propagation and label propagation within a Graph Transformer to make the semi-supervised classification. Furthermore, we propose a masked label prediction method to supervised training our model, preventing it from overfitting in self-loop label information. Experimental results show that UniMP outperforms the previous state-of-the-art models on three main OGBN datasets: *ogbn-products*, *ogbn-proteins* and *ogbn-arxiv* by a large margin, and ablation studies demonstrate the effectiveness of unifying feature propagation and label propagation.

References

[Battaglia *et al.*, 2018] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[Chen *et al.*, 2020] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. *arXiv preprint arXiv:2007.02133*, 2020.

[Chiang *et al.*, 2019] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *SIGKDD*, pages 257–266, 2019.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Duvenaud *et al.*, 2015] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, pages 2224–2232, 2015.

[Gong *et al.*, 2016] Chen Gong, Dacheng Tao, Wei Liu, Liu Liu, and Jie Yang. Label propagation via teaching-to-learn and learning-to-teach. *IEEE TNNLS*, 28(6):1452–1465, 2016.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864, 2016.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[Hu *et al.*, 2020] 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. *arXiv preprint arXiv:2005.00687*, 2020.

[Karasuyama and Mamitsuka, 2013] Masayuki Karasuyama and Hiroshi Mamitsuka. Manifold-based similarity adaptation for label propagation. In *NIPS*, pages 1547–1555, 2013.

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

[Klicpera *et al.*, 2018] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[Li *et al.*, 2018] Qimai Li, Zhichao Han, and Xiaoming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *Neural Computation*, pages 3538–3545, 2018.

[Li *et al.*, 2019] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, pages 9267–9276, 2019.

[Li *et al.*, 2020] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

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

[Liu *et al.*, 2019] 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: Learning*, 2019.

[Liu *et al.*, 2020a] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *SIGKDD*, pages 338–348, 2020.

[Liu *et al.*, 2020b] Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. Bandit samplers for training graph neural networks. *arXiv preprint arXiv:2006.05806*, 2020.

[Niepert *et al.*, 2016] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

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

[Wang and Leskovec, 2019] Hongwei Wang and Jure Leskovec. Unifying graph convolutional neural networks and label propagation. *arXiv: Learning*, 2019.

[Wang and Zhang, 2007] Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. *IEEE TKDE*, 20(1):55–67, 2007.

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

[Xu *et al.*, 2018b] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pages 5453–5462, 2018.

[Yang *et al.*, 2016] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48. PMLR, 2016.

[Zeng *et al.*, 2019] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[Zhang and Lee, 2007] Xinhua Zhang and Wee S Lee. Hyperparameter learning for graph based semi-supervised learning algorithms. In *NIPS*, pages 1585–1592, 2007.

[Zhang *et al.*, 2018] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

[Zhu *et al.*, 2003] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, pages 912–919, 2003.