PERSONALIZED EMBEDDING PROPAGATION: COMBINING NEURAL NETWORKS ON GRAPHS WITH PERSONALIZED PAGERANK

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural message passing algorithms for semi-supervised classification on graphs have recently achieved great success. However, these methods only consider nodes that are a few propagation steps away and the size of this utilized neighborhood cannot be easily extended. In this paper, we use the relationship between graph convolutional networks (GCN) and PageRank to derive an improved propagation scheme based on personalized PageRank. We utilize this propagation procedure to construct personalized embedding propagation (PEP) and its approximation, PEP_A. Our model's training time is on par or faster and its number of parameters on par or lower than previous models. It leverages a large, adjustable neighborhood for classification and can be combined with any neural network. We show that this model outperforms several recently proposed methods for semi-supervised classification on multiple graphs in the most thorough study done so far for GCN-like models.

1 INTRODUCTION

Graphs are ubiquitous in the real world and its description through scientific models. They are used to study the spread of information, to optimize delivery, to recommend new books, to suggest friends, or to find a party's potential voters for the next election. Deep learning approaches have achieved great success on many important graph problems such as link prediction (Grover & Leskovec, 2016; Bojchevski et al., 2018), graph classification (Duvenaud et al., 2015; Niepert et al., 2016; Gilmer et al., 2017) and semi-supervised node classification (Yang et al., 2016; Kipf & Welling, 2017).

Deep learning algorithms on graphs use neural networks to learn node embeddings (representations) that condense the relevant information in a low dimensional vector. In general, deep learning approaches on graphs can be classified into spectral graph convolutional neural networks (Defferrard et al., 2016; Bruna et al., 2014), message passing or neighbor aggregation algorithms (Kipf & Welling, 2017; Hamilton et al., 2017; Pham et al., 2017; Monti et al., 2017; Gilmer et al., 2017; Kearnes et al., 2016), and random walk-based methods (Perozzi et al., 2014; Tang et al., 2015; Nandanwar & Murty, 2016; Grover & Leskovec, 2016). Among these categories, the class of message passing algorithms has garnered particular attention recently. Several works are aimed at improving the basic neighborhood aggregation scheme by using attention mechanisms (Kearnes et al., 2016; Hamilton et al., 2017; Veličković et al., 2018), random walks (Abu-El-Haija et al., 2018a; Ying et al., 2018; Li et al., 2018), edge features (Kearnes et al., 2016; Gilmer et al., 2017; Schlichtkrull et al., 2018) and making it more scalable on large graphs (Chen et al., 2018; Ying et al., 2018). However, all these methods only use the information of a very limited neighborhood for each node.

Increasing the size of the neighborhood used by the algorithm, i.e. its range, however, is not trivial since neighborhood aggregation in this scheme is essentially a type of Laplacian smoothing and too many layers lead to oversmoothing (Li et al., 2018). Xu et al. (2018) highlighted the same problem by establishing a relationship between the message passing algorithm termed Graph Convolutional Network (GCN) by Kipf & Welling (2017) and a random walk. Using this relationship we can see that GCN converges to this random walk's limit distribution as the number of layers increases. This limit distribution is a property of the graph as a whole and does not take the random walk's starting

(root) node into account. As such it is unsuited to describe the root node's neighborhood. Hence, the quality of this aggregation procedure necessarily deteriorates for a high number of layers (or aggregation/propagation steps) since we approach the limit distribution.

To solve this issue, in this paper, we first highlight the inherent connection between the limit distribution and PageRank (Page et al., 1998). We then propose an algorithm that utilizes a propagation scheme derived from *personalized* PageRank instead. This algorithm adds a chance of teleporting back to the root node, which ensures that the score encodes the local neighborhood for every root node (Page et al., 1998). The teleport probability allows us to balance the needs of preserving locality (i.e. staying close to the root node to avoid oversmoothing) and leveraging the information from a large neighborhood. We show that this propagation scheme permits the use of far more (in fact, infinitely many) propagation steps without leading to oversmoothing.

Moreover, while propagation and classification are inherently intertwined in message passing schemes, our proposed algorithm *separates* the neural network from the propagation scheme. This allows us to achieve a much higher range without changing the neural network, whereas in the message passing scheme every additional propagation step would require an additional layer. It also permits the independent development of the propagation algorithm and the neural network that embeds the node features. That is, we can combine any state-of-the-art method for embedding generation with our propagation scheme. We even found that adding our propagation scheme significantly improves the accuracy of networks that have been trained without using any graph information.

Our model achieves state-of-the-art results while requiring fewer parameters and less training time compared to most competing models, with a computational complexity that is linear in the number of edges. We show these results in the most thorough study (including significance testing) of message passing models using graphs with text-based features that has been done so far.

2 Graph convolutional networks and their limited range

We first introduce our notation and explain the problem our model solves. Let $G = (V, E)$ be a graph with nodes V and edges E. Let n denote the number of nodes and m the number of edges. The nodes are described by the feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, with the number of features f per node, and the class (or label) matrix $\boldsymbol{Y} \in \mathbb{R}^{n \times c}$, with the number of classes c. The graph G is described by the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n$ denotes the adjacency matrix with added self-loops.

One simple and widely used message passing algorithm for semi-supervised classification is the Graph Convolutional Network (GCN). In the case of two message passing layers its equation is

$$\boldsymbol{Z}_{\text{GCN}} = \text{softmax}\left(\hat{\tilde{\boldsymbol{A}}}\ \text{ReLU}\left(\hat{\tilde{\boldsymbol{A}}}\boldsymbol{X}\boldsymbol{W}_0\right)\boldsymbol{W}_1\right), \tag{1}$$

where $\boldsymbol{Z} \in \mathbb{R}^{n \times c}$ are the predicted node labels, $\hat{\tilde{\boldsymbol{A}}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ is the symmetrically normalized adjacency matrix with self-loops, with the diagonal degree matrix $\tilde{\boldsymbol{D}}_{ij} = \sum_k \tilde{\boldsymbol{A}}_{ik}\delta_{ij}$, and $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ are trainable weight matrices (Kipf & Welling, 2017).

With two GCN-layers, only neighbors in the two-hop neighborhood are considered. There are essentially two reasons why a message passing algorithm like GCN can't be trivially expanded to use a larger neighborhood. First, aggregation by averaging causes oversmoothing if too many layers are used. It, therefore, loses its focus on the local neighborhood (Li et al., 2018). Second, most common aggregation schemes use learnable weight matrices in each layer. Therefore, using a larger neighborhood necessarily increases the depth and number of learnable parameters of the neural network (the second aspect can be circumvented by using weight sharing, which is typically not the case, though). However, the required neighborhood size and neural network depth are two completely orthogonal aspects. This fixed relationship is a strong limitation and leads to bad compromises.

We will start by concentrating on the first issue. Using a randomization assumption for the ReLU activations, Xu et al. (2018) have shown that for a k-layer GCN the influence score of node x on y, $I(x, y) = \sum_i \sum_j \frac{\partial \boldsymbol{Z}_{yi}}{\partial \boldsymbol{X}_{xj}}$, is proportional in expectation to a slightly modified k-step random walk distribution starting at the root node x, $P_{\text{rw'}}(x \rightarrow y, k)$. Hence, the information of node x spreads to node y in a random walk-like manner. If we take the limit $k \rightarrow \infty$ and the graph is irreducible and aperiodic, this random walk

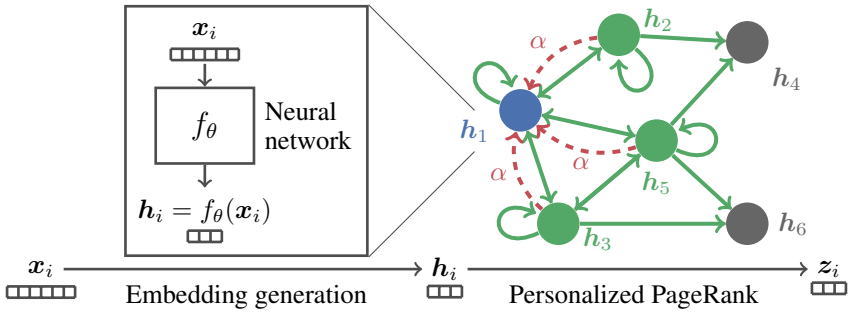

Figure 1: Illustration of (approximate) personalized embedding propagation (PEP, PEP_A). Embeddings are first calculated for each node independently by a neural network. These embeddings are then propagated using an adaptation of personalized PageRank.

probability distribution $P_{\text{rw}'}(x \rightarrow y, k)$ converges to the limit (or stationary) distribution $P_{\text{lim}}(\rightarrow y)$. This distribution can be obtained by solving the equation $\boldsymbol{\pi}_{\text{lim}} = \hat{\tilde{\boldsymbol{A}}}\boldsymbol{\pi}_{\text{lim}}$. Obviously, the result only depends on the graph as a whole and is independent of the random walk's starting (root) node x. This global property is therefore unsuitable for describing the root node's neighborhood.

3 Personalized embedding propagation (PEP)

From message passing to personalized PageRank. We can solve the problem of lost focus by recognizing the connection between the limit distribution and PageRank (Page et al., 1998). The only differences between these two are the added self-loops and the adjacency matrix normalization in $\hat{\tilde{\boldsymbol{A}}}$. Original PageRank is calculated via $\boldsymbol{\pi}_{\text{pr}} = \boldsymbol{A}_{\text{rw}}\boldsymbol{\pi}_{\text{pr}}$, with $\boldsymbol{A}_{\text{rw}} = \boldsymbol{A}\boldsymbol{D}^{-1}$. Having made this connection we can now consider using a variant of PageRank that takes the root node into account – personalized PageRank (Page et al., 1998). We define the root node x via the teleport vector $\boldsymbol{i}_x$, which is a one-hot indicator vector. Our adaptation of personalized PageRank can be obtained for node x using the recurrent equation $\boldsymbol{\pi}_{\text{ppr}}(\boldsymbol{i}_x) = (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\boldsymbol{\pi}_{\text{ppr}}(\boldsymbol{i}_x) + \alpha\boldsymbol{i}_x$, with the teleport (or restart) probability $\alpha \in (0, 1]$. By solving this equation, we obtain

$$\boldsymbol{\pi}_{\text{ppr}}(\boldsymbol{i}_x) = \alpha\left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\right)^{-1}\boldsymbol{i}_x. \tag{2}$$

Introducing the teleport vector $\boldsymbol{i}_x$ allows us to preserve the node's local neighborhood even in the limit distribution. In this model the influence score of root node x on node y, $I(x, y)$, is proportional to the y-th element of our personalized PageRank $\boldsymbol{\pi}_{\text{ppr}}(\boldsymbol{i}_x)$. This value is different for every root node. How fast it decreases as we move away from the root node can be adjusted via α. By substituting the indicator vector $\boldsymbol{i}_x$ with the unit matrix $\boldsymbol{I}_n$ we obtain our fully personalized PageRank matrix $\boldsymbol{\Pi}_{\text{ppr}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}})^{-1}$, whose element (yx) specifies the influence score of node x on node y, $I(x, y) \propto \boldsymbol{\Pi}_{\text{ppr}}^{(yx)}$. Note that due to symmetry $\boldsymbol{\Pi}_{\text{ppr}}^{(yx)} = \boldsymbol{\Pi}_{\text{ppr}}^{(xy)}$, i.e. the influence of x on y is equal to the influence of y on x. Also, this inverse always exists since $\frac{1}{1-\alpha} > 1$ and therefore can't be an eigenvalue of $\hat{\tilde{\boldsymbol{A}}}$ (see Appendix A).

Personalized embedding propagation (PEP). To utilize the above influence scores for semi-supervised classification we can generate embeddings for each node independently and then propagate them according to the fully personalized PageRank scheme to generate the predictions. This is the foundation of personalized embedding propagation. PEP's model equation is

$$\boldsymbol{Z}_{\text{PEP}} = \text{softmax}\left(\alpha\left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\right)^{-1}\boldsymbol{H}\right), \qquad \boldsymbol{H}_{i,:} = f_\theta(\boldsymbol{X}_{i,:}), \tag{3}$$

where $\boldsymbol{X}$ is the feature matrix and f_θ a neural network with parameter set θ generating the embeddings $\boldsymbol{H} \in \mathbb{R}^{n\times c}$. Note that f_θ operates on each node's features independently.

As a consequence, PEP separates the neural network used for generating embeddings from the propagation scheme. This separation additionally solves the second issue mentioned above: the

depth of the neural network is now fully independent of the propagation algorithm. As we saw when connecting GCN to PageRank, personalized PageRank can effectively use even infinitely many neighborhood aggregation layers, which is clearly not possible in the classical message passing framework. Furthermore, the separation gives us the flexibility to use any method for embedding generation, e.g. deep convolutional neural networks for graphs of images.

While embedding generation and propagation happen consecutively during inference, it is important to note that **the model is trained end-to-end**. That is, the gradient flows through the propagation scheme during backpropagation (implicitly considering infinitely many neighborhood aggregation layers). Adding these propagation effects significantly improves the model's accuracy.

Efficiency analysis. Directly calculating the fully personalized PageRank matrix $\mathbf{\Pi}_{\text{ppr}}$, is computationally inefficient and results in a dense $\mathbb{R}^{n\times n}$ matrix. Using this matrix would lead to a computational complexity and memory requirement of $\mathcal{O}(n^2)$ for training and inference.

To solve this issue, reconsider the equation $\boldsymbol{Z} = \alpha(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}})^{-1}\boldsymbol{H}$. Instead of viewing this equation as a combination of a dense fully *personalized* PageRank matrix with the node embedding matrix, we can also view it as a variant of *topic-sensitive* PageRank, where every *column* of $\boldsymbol{H}$ defines an (unnormalized) distribution over nodes that acts as a teleport set (Haveliwala, 2002). A high value in this set/column will increase the node's final topic-sensitive PageRank. Hence, these teleport sets are directly connected to the result $\boldsymbol{Z}$, i.e. the class predictions. Considering the (columns of the) embedding matrix as (unnormalized) teleport sets enables us to approximate PEP via an approximate computation of topic-sensitive PageRank.

PEP$_{\text{A}}$. More precisely, approximate personalized embedding propagation (PEP$_{\text{A}}$) achieves linear computational complexity by approximating the topic-sensitive PageRank via power iteration. While the power iteration of PageRank is connected to the regular random walk, the power iteration of topic-sensitive PageRank is directly related to a random walk with restarts. Each power iteration (random walk/propagation) step of our topic-sensitive PageRank variant is, thus, calculated via

$$\begin{aligned}
\boldsymbol{Z}^{(0)} &= \boldsymbol{H} = f_\theta(\boldsymbol{X}),\\
\boldsymbol{Z}^{(k+1)} &= (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\boldsymbol{Z}^{(k)} + \alpha\boldsymbol{H},\\
\boldsymbol{Z}^{(K)} &= \text{softmax}\left((1-\alpha)\hat{\tilde{\boldsymbol{A}}}\boldsymbol{Z}^{(K-1)} + \alpha\boldsymbol{H}\right),
\end{aligned} \tag{4}$$

where the embedding matrix $\boldsymbol{H}$ acts as both the starting vector and the teleport set, K defines the number of power iteration steps and $k \in [0, K-1]$. Note that this method retains the graph's sparsity and never constructs an $\mathbb{R}^{n\times n}$ matrix. The convergence of this iterative scheme can be shown by investigating the resulting series (see Appendix B).

Note that the propagation scheme of this model does not require any additional parameters to train – as opposed to models like GCN, which require more parameters for each additional propagation layer. We can therefore propagate very far with very few parameters. Our experiments show that this ability is indeed very beneficial (Section 6). A similar model expressed in the message passing framework would therefore not be able to achieve the same level of performance.

In both PEP and PEP$_{\text{A}}$, the size of the neighborhood influencing each node can be adjusted via the teleport probability α. The freedom to choose α allows us to adjust the model for different types of networks, since varying graph types require the consideration of different neighborhood sizes, as shown in Section 6 and described by Grover & Leskovec (2016) and Abu-El-Haija et al. (2018b).

4 Related work

Several works have tried to improve the training of message passing algorithms and increase the neighborhood available at each node by adding skip connections (Li et al., 2016; Pham et al., 2017; Hamilton et al., 2017; Ying et al., 2018). One recent approach combined skip connection with aggregation schemes (Xu et al., 2018). However, the range of all of these models is still limited, as apparent in the low number of message passing layers used. While it is possible to add skip connections in the neural network used by our algorithm, this would not influence the propagation scheme. Our approach to solving the range problem is therefore unrelated to these models.

Table 1: Dataset statistics

Dataset	Type	Classes	Features	Nodes	Edges	Label rate
CITESEER	Citation	6	3703	2110	3668	0.036
CORA-ML	Citation	7	2879	2810	7981	0.047
PUBMED	Citation	3	500	19 717	44 324	0.003
MICROSOFT ACADEMIC	Co-author	15	6805	18 333	81 894	0.016

Li et al. (2018) facilitated training by combining message passing with co- and self-training. The improvements achieved by this combination are similar to results reported with other semi-supervised classification models (Buchnik & Cohen, 2018). Note that most algorithms, including ours, can be improved using self- and co-training. However, each additional step used by these methods corresponds to a full training cycle and therefore significantly increases the training time.

Another direction of research enhances neighborhood aggregation with attention mechanisms (Kearnes et al., 2016; Hamilton et al., 2017; Veličković et al., 2018). While these methods show great promise, especially on chemical graphs, please note that this approach is orthogonal to ours. Our method extends the considered neighborhood, it does not (re-)weight the available connections.

5 EXPERIMENTAL SETUP

Recently, many experimental evaluations have suffered from overfitting by only using a single training/validation/test split, by not distinguishing clearly between the validation and test set, and by finetuning hyperparameters to each dataset or even to each data split. Message-passing algorithms are very sensitive to both data splits and weight initialization (as clearly shown by our evaluation). Thus a carefully designed evaluation protocol is extremely important. Our work aims to establish such a thorough evaluation protocol. First, we run each experiment 100 times on multiple random splits and initializations. Second, we split the data into a visible and a test set, which do not change. The test set was only used *once* to evaluate the final performance; and in particular, has never been used to perform hyperparameter and model selection. Furthermore, since our datasets are somewhat similar, we use the *same* hyperparameters across datasets. To prevent experimental bias we have optimized the hyperparameters of *all* baselines in the same setup using a grid search on CITESEER and CORA-ML and use the *same* early stopping criterion across models.

Finally, to ensure the robustness of our experimental setup, we calculate confidence intervals via bootstrapping and report the p-values of a paired t-test for our main claims. To our knowledge, this is the most thorough study on GCN-like models which has been done so far on graphs having text-based features. More details about the experimental setup are provided in Appendix C.

Datasets. We use four text-classification datasets for evaluation. CITESEER (Sen et al., 2008), CORA-ML (McCallum et al., 2000) and PUBMED (Namata et al., 2012) are citation graphs, where each node represents a paper and the edges represent citations between them. We further introduce the MICROSOFT ACADEMIC graph, where edges represent co-authorship. All graphs use a bag-of-words representation of the papers' abstracts as features. Table 1 reports the dataset statistics.

Baseline models. We use five state-of-the-art models as baselines: GCN (Kipf & Welling, 2017), network of GCNs (N-GCN) (Abu-El-Haija et al., 2018a), graph attention networks (GAT) (Veličković et al., 2018), bootstrapped (self-trained) feature propagation (bt. FP) (Buchnik & Cohen, 2018) and jumping knowledge networks with concatenation (JK) (Xu et al., 2018). For GCN we also show the results of the (unoptimized) vanilla version (V. GCN) to demonstrate the strong impact of early stopping and hyperparameter optimization. We describe the optimized hyperparameters of all models in Appendix D.

Model hyperparameters. To ensure a fair model comparison we used a neural network for PEP that is structurally very similar to GCN and has the same number of parameters. We use two layers with $h = 64$ hidden units. We apply L_2 regularization with $\lambda = 0.005$ on the weights of the first layer and use dropout with dropout rate $d = 0.5$ on both layers and the adjacency matrix. For PEP_A, adjacency dropout is resampled for each power iteration step. For propagation we use the teleport probability $\alpha = 0.1$ and $K = 10$ power iteration steps for PEP_A. Furthermore, the combination of

Table 2: Average accuracy with uncertainties calculated by bootstrapping with 95 % confidence level. PEP and PEP_A achieve the highest accuracy on most of the investigated datasets.

Model	CITESEER	CORA-ML	PUBMED	MICROSOFT
V. GCN	73.51 ± 0.48	82.30 ± 0.34	77.65 ± 0.40	91.65 ± 0.09
GCN	75.40 ± 0.30	83.41 ± 0.39	78.68 ± 0.38	92.10 ± 0.08
N-GCN	74.25 ± 0.40	82.25 ± 0.30	77.43 ± 0.42	$\mathbf{92.86 \pm 0.11}$
GAT	75.39 ± 0.27	84.37 ± 0.24	77.76 ± 0.44	91.22 ± 0.07
JK	73.03 ± 0.47	82.69 ± 0.35	77.88 ± 0.38	91.71 ± 0.10
Bt. FP	73.55 ± 0.57	80.84 ± 0.97	72.94 ± 1.00	91.61 ± 0.24
PEP	$\mathbf{75.83 \pm 0.27}$	$\mathbf{85.29 \pm 0.25}$	-	-
PEP_A	75.73 ± 0.30	85.09 ± 0.25	$\mathbf{79.73 \pm 0.31}$	92.60 ± 0.08

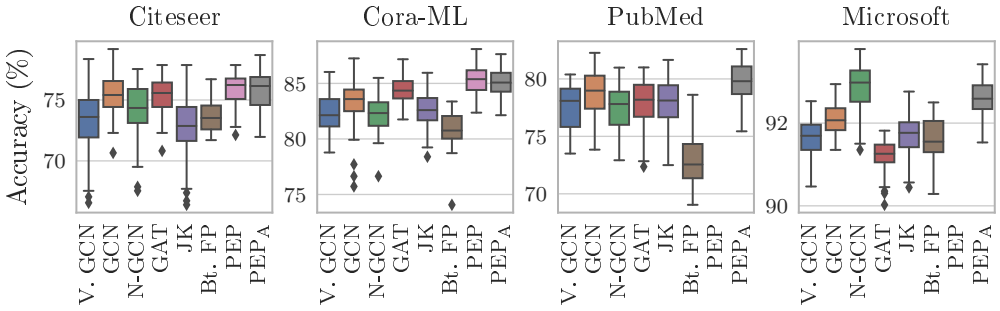

Figure 2: Accuracy distributions of different models. The high standard deviation requires a statistically thorough model evaluation.

this shallow neural network with a comparatively high number of power iteration steps achieved the best results during hyperparameter optimization.

6 RESULTS

Overall accuracy. The results for the accuracy (micro F1-score) are summarized in Table 2. Similar trends are observed for the macro F1-score (see Appendix E). Both PEP and PEP_A outperform the baseline models on all datasets except MICROSOFT ACADEMIC. This result is statistically significant $p < 0.05$, as tested via a paired t-test (see Appendix F). N-GCN outperforming PEP can be attributed to its high number of trainable parameters. N-GCN has 5 times as many parameters, which is a large advantage on this graph since it has a very high number of features. This becomes apparent when we reduce the number of training samples and PEP surpasses N-GCN since the latter doesn't have enough information available to effectively train its parameters (see Appendix G). The table furthermore shows that the advantages reported by recent works shrink when the early stopping criterion is harmonized, hyperparameters are properly optimized and multiple data splits are considered. A simple GCN with optimized hyperparameters outperforms several other recently proposed models in our setup.

Figure 2 shows how broad the accuracy distribution of each model is. This is caused by both the random initialization and the different data splits (train / early stopping / test). This shows how crucial a statistically thorough evaluation is for a conclusive model comparison. Moreover, it shows the sensitivity (robustness) of each method, e.g. PEP, PEP_A and GAT typically have lower variance.

Table 3: Average training time per epoch. PEP and PEP_A are only slightly slower than GCN and much faster than more sophisticated methods like GAT.

Graph	V. GCN	GCN	N-GCN	GAT	JK	Bt. FP	PEP	PEP_A
CITESEER	37.6 ms	35.3 ms	115.9 ms	187.0 ms	57.5 ms	-	49.2 ms	43.3 ms
CORA-ML	32.4 ms	36.5 ms	118.9 ms	217.4 ms	43.6 ms	-	55.3 ms	42.7 ms
PUBMED	48.6 ms	48.3 ms	342.6 ms	1029.8 ms	77.8 ms	-	-	64.1 ms
MICROSOFT	45.5 ms	39.2 ms	328.5 ms	772.2 ms	61.9 ms	-	-	56.3 ms

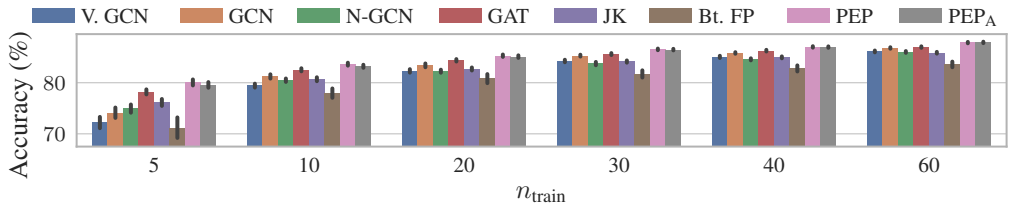

Figure 3: Accuracy for different training set sizes (number of labeled nodes per class) on CORA-ML. PEP's dominance increases further for smaller training set sizes.

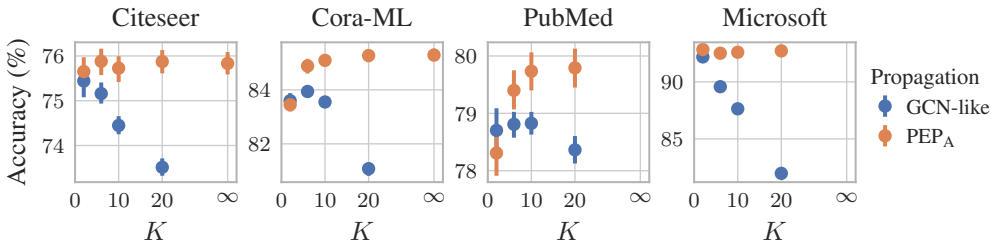

Figure 4: Accuracy depending on the number of propagation steps K. The accuracy breaks down for the GCN-like propagation ($\alpha = 0$), while it increases and stabilizes when using PEP$_A$ ($\alpha = 0.1$).

Training time per epoch. We report the average training time per epoch in Table 3. We decided to only compare the training time per epoch since all hyperparameters were solely optimized for accuracy and the used early stopping criterion is very generous. Obviously, (exact) PEP can only be applied to moderately sized graphs, while PEP$_A$ scales to large data. On average, PEP$_A$ is around $25\,\%$ slower than GCN due to its higher number of matrix multiplications. It scales similarly with graph size as GCN and is therefore significantly faster than other more sophisticated models like GAT. This is observed even though our implementation improved GAT's training time roughly by a factor of 2 compared to the reference implementation.

Bootstrapped FP does not use regular training. We observed that it is comparatively fast for small graphs with few features. However, sparse feature matrices become dense in this model and cause the method to be very slow on graphs with a high number of features.

Training set size. Since the labeling rate is often very small for real world datasets, investigating how the models perform with a small number of training samples is very important. Figure 3 shows how the number of training nodes per class n_{train} impacts the achieved accuracy on CORA-ML. The results on the other datasets can be found in Appendix G. The dominance of PEP and PEP$_A$ becomes specifically clear for small training sets. This can be attributed to their higher range, which allows them to better propagate the information further away from the (few) training nodes.

Number of power iteration steps. Figure 4 shows how the accuracy depends on the number of power iteration steps for two different propagation schemes. The first mimics the standard propagation as known from GCNs (i.e. taking $\alpha = 0$ in PEP$_A$). As clearly shown the performance breaks down as we increase the number of power iteration steps K (since we approach the global PageRank solution). However, when using the power of personalized propagation (here with $\alpha = 0.1$) the accuracy *increases* and converges to the solution of (exact) PEP with infinitely many propagation steps, thus demonstrating the personalized propagation principle is indeed beneficial. As also shown in the figure, it is enough to use a moderate number of power iterations (e.g. $K = 10$) to approximate the exact PEP solution.

Teleport probability α. Figure 5 shows how the accuracy on the validation set is affected by the hyperparameter α. While the optimum differs slightly for every dataset, we consistently found a teleport probability of around $\alpha \in [0.05, 0.2]$ to perform best. In real-world scenarios this probability should be adjusted for the dataset under investigation, since different graphs exhibit different neighborhood structures (Grover & Leskovec, 2016; Abu-El-Haija et al., 2018b).

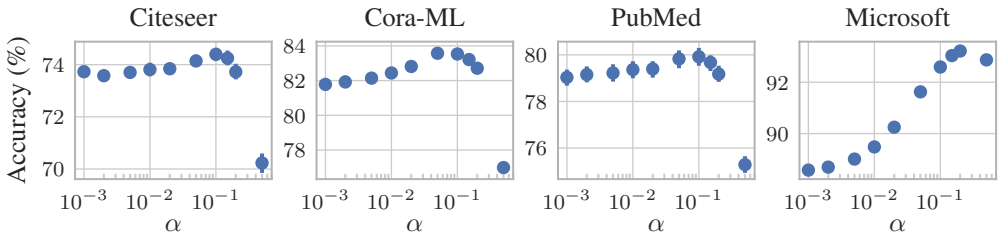

Figure 5: Accuracy depending on teleport probability α. The optimum typically lies within $\alpha \in [0.05, 0.2]$, but changes for different types of datasets.

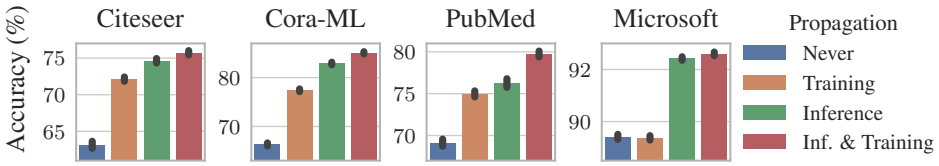

Figure 6: Accuracy of PEP_A with propagation used only during training/inference. The best results are achieved with full propagation, but only using propagation during inference achieves decent results as well.

Neural network without propagation. PEP and PEP_A are trained end-to-end, with the propagation scheme affecting (i) the neural network f_θ during training, and (ii) the classification decision during inference. Investigating how the model performs without propagation shows if and how valuable this addition is. Figure 6 shows how propagation affects both training and inference. "Never" denotes the case where no propagation is used; essentially we train and apply a standard multilayer perceptron (MLP) f_θ using the features only. "Training" denotes the case where we use PEP_A during training to learn f_θ; at inference time, however, only f_θ is used to predict the class labels. "Inference", in contrast, denotes the case where f_θ is trained without PEP_A (i.e. standard MLP on features); this pretrained network with fixed weights, however, is then used within PEP_A during inference time. Finally, "Inf. & Training" denotes the regular PEP_A, which always uses propagation.

The best results are achieved with regular PEP_A, which validates our approach. However, on most datasets the classification accuracy decreases surprisingly little when propagation is only performed during inference. Skipping propagation during training can significantly reduce training time for large graphs, since all nodes can be handled independently. Furthermore, this shows that our model can be combined with pretrained neural networks that do not incorporate any graph information and still significantly improve their accuracy. Moreover, Figure 6 shows that only applying propagation during training can also lead to large improvements. This suggests that our model should be able to handle cases of online and inductive learning where only the features and not the neighborhood information of an incoming (previously unobserved) node are available.

7 Conclusion

In this paper we have introduced personalized embedding propagation (PEP) and a fast approximation, PEP_A. We derived this model by considering the relationship between GCN and PageRank and extending it to personalized PageRank. This model solves the limited range problem inherent in many message passing models without introducing any additional parameters. It uses the information from a large, adjustable neighborhood for classifying each node and is computationally efficient. This model outperforms several modern methods for semi-supervised classification on multiple graphs in the most thorough study which has been done for GCN-like models so far.

For future work we want to combine PEP with more complex neural networks used e.g. in computer vision or natural language processing. Furthermore, we want to look into faster or incremental approximations of personalized PageRank (Bahmani et al., 2010; 2011; Lofgren et al., 2014) and investigate more sophisticated propagation schemes.

REFERENCES

Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification. In *International Workshop on Mining and Learning with Graphs (MLG)*, 2018a.

Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. Watch Your Step: Learning Node Embeddings via Graph Attention. In *NIPS*, 2018b.

Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast Incremental and Personalized PageRank. *VLDB*, 4(3):173–184, 2010.

Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. Fast Personalized PageRank on MapReduce. In *SIGMOD*, pp. 973–984, 2011.

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating Graphs via Random Walks. In *ICML*, pp. 609–618, 2018.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Deep Locally Connected Networks on Graphs. *ICLR*, 2014.

Eliav Buchnik and Edith Cohen. Bootstrapped Graph Diffusions: Exposing the Power of Nonlinearity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2 (1):1–19, April 2018.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *ICLR*, 2018.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*, pp. 3837–3845, 2016.

David K. Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NIPS*, pp. 2224–2232, 2015.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *ICML*, pp. 1263–1272, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *KDD*, pp. 855–864, 2016.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NIPS*, pp. 1025–1035, 2017.

Taher H. Haveliwala. Topic-sensitive PageRank. In *WWW*, pp. 517–526, 2002.

Steven M. Kearnes, Kevin McCloskey, Marc Berndl, Vijay S. Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *ICLR*, 2015.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *ICLR*, 2017.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI*, 2018.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated Graph Sequence Neural Networks. In *ICLR*, 2016.

Peter Lofgren, Siddhartha Banerjee, Ashish Goel, and Seshadhri Comandur. FAST-PPR: scaling personalized pagerank estimation for large graphs. In *KDD*, pp. 1436–1445, 2014.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *CVPR*, pp. 5425–5434, 2017.

Galileo Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven Active Surveying for Collective Classification. In *International Workshop on Mining and Learning with Graphs (MLG)*, pp. 8, 2012.

Sharad Nandanwar and M. N. Murty. Structural Neighborhood Based Classification of Nodes in a Network. In *KDD*, pp. 1085–1094, 2016.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *ICML*, pp. 2014–2023, 2016.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford InfoLab*, 1998.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: online learning of social representations. In *KDD*, pp. 701–710, 2014.

Trang Pham, Truyen Tran, Dinh Q. Phung, and Svetha Venkatesh. Column Networks for Collective Classification. In *AAAI*, pp. 2485–2491, 2017.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In *Extended Semantic Web Conference (ESWC)*, pp. 593–607, 2018.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 29(3):93–106, 2008.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *WWW*, pp. 1067–1077, 2015.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *ICLR*, 2018.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*, pp. 5449–5458, 2018.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *ICML*, pp. 40–48, 2016.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *KDD*, 2018.

A EXISTENCE OF $\boldsymbol{\Pi}_{\text{PPR}}$

The matrix

$$\boldsymbol{\Pi}_{\text{ppr}} = \alpha \left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}} \right)^{-1} \tag{5}$$

exists iff the determinant $\det(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}}) \neq 0$, which is the case iff $\det(\hat{\tilde{\boldsymbol{A}}} - \frac{1}{1-\alpha}\boldsymbol{I}_n) \neq 0$, i.e. iff $\frac{1}{1-\alpha}$ is not an eigenvalue of $\hat{\tilde{\boldsymbol{A}}}$. This value is always larger than 1 since the teleport probability $\alpha \in (0, 1]$. Furthermore, the symmetrically normalized matrix $\hat{\tilde{\boldsymbol{A}}}$ has the same eigenvalues as the row-stochastic matrix $\tilde{\boldsymbol{A}}_{\text{rw}}$. This can be shown by multiplying the eigenvalue equation $\hat{\tilde{\boldsymbol{A}}}\boldsymbol{v} = \lambda\boldsymbol{v}$ with $\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ from left and substituting $\boldsymbol{w} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{v}$. However, the largest eigenvalue of a row-stochastic matrix is 1, as can be proven using the Gershgorin circle theorem. Hence, $\frac{1}{1-\alpha}$ can't be an eigenvalue and $\boldsymbol{\Pi}_{\text{ppr}}$ always exists.

B CONVERGENCE OF PEP_{A}

PEP_{A} uses the iterative equation

$$\boldsymbol{Z}^{(k+1)} = (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\boldsymbol{Z}^{(k)} + \alpha\boldsymbol{H}. \tag{6}$$

After the k-th propagation step, the resulting embeddings are

$$\boldsymbol{Z}^{(k)} = \left((1-\alpha)^k \hat{\tilde{\boldsymbol{A}}}^k + \alpha \sum_{i=0}^{k-1} (1-\alpha)^i \hat{\tilde{\boldsymbol{A}}}^i \right) \boldsymbol{H}. \tag{7}$$

If we take the limit $k \to \infty$ the left term tends to 0 and the right term becomes a geometric series. The series converges since $\alpha \in (0, 1]$ and $\hat{\tilde{\boldsymbol{A}}}$ is symmetrically normalized and therefore $\det(\hat{\tilde{\boldsymbol{A}}}) \leq 1$, resulting in

$$\boldsymbol{Z}^{(\infty)} = \alpha \left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}} \right)^{-1} \boldsymbol{H}, \tag{8}$$

which is the equation for calculating (exact) PEP.

C EXPERIMENTAL DETAILS

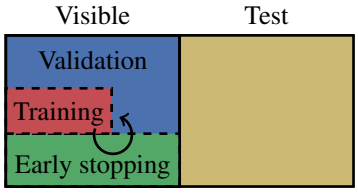

Figure 7: Illustration of the node sampling procedure.

The sampling procedure is illustrated in Figure 7. The data is first split into a visible and a test set. For the visible set 1500 nodes were sampled for the citation graphs and 5000 for MICROSOFT ACADEMIC. The test set contains all remaining nodes. We use three different label sets in each experiment: A training set of 20 nodes per class, an early stopping set of 500 nodes and either a validation or test set. The validation set contains the remaining nodes of the visible set. We use 20 random seeds for determining the splits. These seeds are drawn once and fixed across runs to facilitate comparisons. We use one set of seeds for the validation splits and a different set for the test splits. Each experiment is run with 5 random initializations on each data split, leading to a total of 100 runs per experiment.

The early stopping criterion uses a patience of $p = 100$ and an (unreachably high) maximum of $n = 10\,000$ epochs. The patience is reset whenever either the accuracy increases or the loss decreases on the early stopping set. We choose the parameter set achieving the highest accuracy and break ties by selecting the lowest loss on this set. This early stopping strategy was inspired by GAT (Veličković et al., 2018).

We used TensorFlow (Martín Abadi et al., 2015) for all experiments except bootstrapped feature propagation. All uncertainties and confidence intervals correspond to a confidence level of $95\,\%$ and were calculated by bootstrapping with 1000 samples.

We use the Adam optimizer with a learning rate of $l = 0.01$ and cross-entropy loss for all models (Kingma & Ba, 2015). Weights are initialized as described in Glorot & Bengio (2010). The feature matrix is L_1 normalized per row.

D Baseline hyperparameters

Vanilla GCN uses the original settings of two layers with $h = 16$ hidden units, no dropout on the adjacency matrix, L_2 regularization parameter $\lambda = 5 \times 10^{-4}$ and the original early stopping with a maximum of 200 steps and a patience of 10 steps based on the loss.

The optimized GCN uses two layers with $h = 64$ hidden units, dropout on the adjacency matrix with $d = 0.5$ and L_2 regularization parameter $\lambda = 0.02$.

N-GCN uses $h = 16$ hidden units, $R = 4$ heads per random walk length and random walks of up to $K - 1 = 4$ steps. It uses L_2 regularization on all layers with $\lambda = 1 \times 10^{-5}$ and the attention variant for merging the predictions (Abu-El-Haija et al., 2018a). Note that this model effectively uses $RKh = 320$ hidden units, which is 5 times as many as the optimized GCN model, GAT and PEP use.

For GAT we use the (well optimized) original hyperparameters, except the L_2 regularization parameter $\lambda = 0.001$ and learning rate $l = 0.01$. As opposed to the original paper, we do not use different hyperparameters on PubMed, as described in our experimental setup.

Bootstrapped feature propagation uses a return probability of $\alpha = 0.2$, 10 propagation steps, 10 bootstrapping (self-training) steps with $r = 0.1n$ training nodes added per step. We add the training nodes with the lowest entropy on the perdictions. The number of nodes added per class is based on the class proportions estimated by the predictions. Note that this model does not include any stochasticity in its initialization. We therefore only run it once per train/early stopping/test split.

For the jumping knowledge networks we use the concatenation variant with three layers and $h = 64$ hidden units per layer. We apply L_2 regularization with $\lambda = 0.001$ on all layers and perform dropout with $d = 0.5$ on all layers but not on the adjacency matrix.

E F1 score

Table 4: Average macro F1 score with uncertainties calculated by bootstrapping with $95\,\%$ confidence level. PEP achieves the highest F1 score on most of the investigated datasets.

Model	Citeseer	Cora-ML	PubMed	Microsoft
V. GCN	0.7002 ± 0.0043	0.8205 ± 0.0027	0.7801 ± 0.0038	0.9000 ± 0.0008
GCN	0.7065 ± 0.0037	0.8289 ± 0.0030	0.7883 ± 0.0032	0.9045 ± 0.0008
N-GCN	0.7021 ± 0.0035	0.8183 ± 0.0024	0.7773 ± 0.0040	$\mathbf{0.9144 \pm 0.0012}$
GAT	0.7062 ± 0.0029	0.8359 ± 0.0025	0.7777 ± 0.0040	0.8917 ± 0.0007
JK	0.6914 ± 0.0043	0.8202 ± 0.0026	0.7799 ± 0.0039	0.8985 ± 0.0012
Bt. FP	0.6789 ± 0.0055	0.8026 ± 0.0082	0.7448 ± 0.0079	0.8997 ± 0.0018
PEP	0.7102 ± 0.0041	$\mathbf{0.8454 \pm 0.0021}$	-	-
PEP_A	$\mathbf{0.7105 \pm 0.0038}$	0.8429 ± 0.0022	$\mathbf{0.7966 \pm 0.0031}$	0.9096 ± 0.0009

F PAIRED t-TEST

Table 5: p-value of the paired t-test with respect to accuracy.

Model	CITESEER	CORA-ML	PUBMED	MICROSOFT
PEP	1.02×10^{-3}	5.96×10^{-14}	-	-
PEP_A	1.19×10^{-2}	4.27×10^{-9}	2.19×10^{-15}	-

Table 6: p-value of the paired t-test with respect to F1 score.

Model	CITESEER	CORA-ML	PUBMED	MICROSOFT
PEP	4.08×10^{-2}	4.50×10^{-14}	-	-
PEP_A	2.32×10^{-2}	1.07×10^{-8}	8.70×10^{-14}	-

G TRAINING SET SIZE

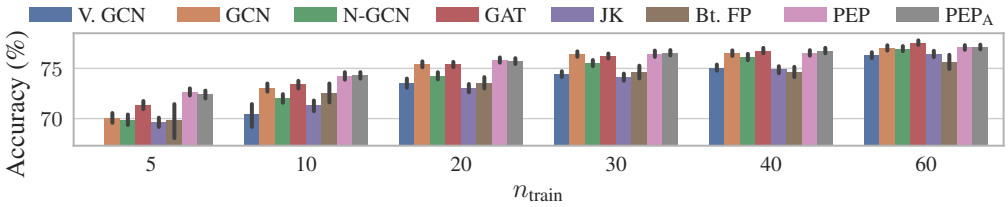

Figure 8: Accuracy for different training set sizes on CITESEER.

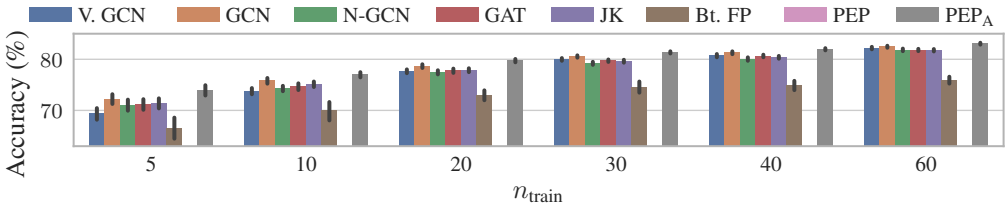

Figure 9: Accuracy for different training set sizes on PUBMED.

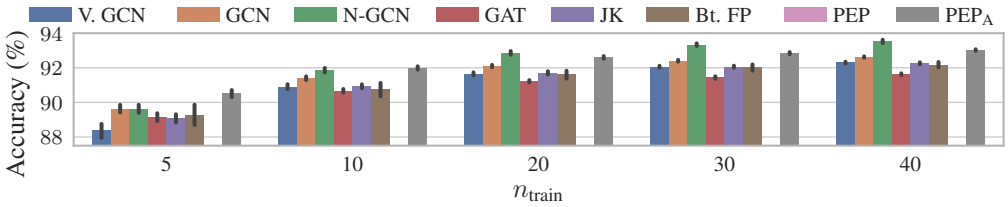

Figure 10: Accuracy for different training set sizes on MICROSOFT ACADEMIC.