Is PageRank All You Need for Scalable Graph Neural Networks?

Aleksandar Bojchevski
a.bojchevski@in.tum.de
Technical University of Munich

Johannes Klicpera
klicpera@in.tum.de
Technical University of Munich

Bryan Perozzi
bperozzi@acm.org
Google AI

Martin Blais
blais@google.com
Google AI

Amol Kapoor
ajkapoor@google.com
Google AI

Michal Lukasik
mlukasik@google.com
Google AI

Stephan Günnemann
guennemann@in.tum.de
Technical University of Munich

ABSTRACT

Graph neural networks (GNNs) have emerged as a powerful approach for solving many network mining tasks. However, efficiently utilizing them on web-scale data remains a challenge despite related advances in research. Most recently proposed scalable GNNs rely on an expensive recursive message-passing procedure to propagate information through the graph. We circumvent this limitation by leveraging connections between GNNs and personalized PageRank and we develop a model that incorporates multi-hop neighborhood information in a single (non-recursive) step. Our work-in-progress approach PPRGo is significantly faster than multi-hop models while maintaining state-of-the-art prediction performance. We demonstrate the strengths and scalability of our approach on graphs orders of magnitude larger than typically considered in the literature.

ACM Reference Format:
Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Martin Blais, Amol Kapoor, Michal Lukasik, and Stephan Günnemann. 2019. Is PageRank All You Need for Scalable Graph Neural Networks?. In *15th International Workshop on Mining and Learning with Graphs, August 05, 2019, Anchorage, AK*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Graphs are a natural way of representing a wide variety of real-life data from sociology, biology, finance, and many other domains. Recently, traditional graph mining techniques [9, 19] have given way to approaches based on (deep) graph neural networks (GNNs) [4, 6, 56, 61, 62] since these show superior performance on a wide variety of network mining tasks including: semi-supervised node classification [23, 28, 49], link prediction [5, 29, 60], community detection [12, 27, 45], graph classification/regression [20, 40, 57], and graph-based recommendation [38, 59].

Following the success of GNNs on academic datasets, there has been increasing interest in scaling up these methods to large graphs [10, 11, 18, 23, 25, 43, 59]. While some advances towards scalable GNNs have been made there are still many open questions. In particular, the scalability of most methods has been demonstrated on graphs smaller than 250K nodes, while in practice we are interested in graphs with (tens of) billions of nodes and edges. Ying et al. [59] is the only work that demonstrates scalability on web-scale graphs of such orders of magnitude. Their focus, however, is on using GNNs for recommendation, while our focus is on scalable node classification. To capture multi-hop neighborhood information their model utilizes an expensive message-passing procedure which is inherently less scalable compared to our approach. Moreover, their model was run on a single machine, while our approach leverages distributed training for further scalablity.

In this paper, we leverage connections between GNNs and personalized PageRank [1, 26] to develop a model that incorporates important neighborhood information without explicit message-passing. Xu et al. [58] and Li et al. [33] study some of these connections and show that the influence of a node i on all other nodes for a k-layer GNN is proportional in expectation to a (modified) k-step random walk distribution of random walks starting at node i. As the number of layers k increases we lose focus of the local neighborhood of a given node and in the limit of an infinitely deep GNN we converge to the stationary distribution of the respective Markov chain making the final representations identical for all nodes. Thus, stacking many graph convolution layers leads to over-smoothing, which makes the representations indistinguishable and prevents learning. Note that in practice we observe the over-smoothing effect after only a few layers [30] as evidenced by the sharp decline in performance with additional layers on common benchmark graphs.

The PPNP model proposed by Klicpera et al. [30] alleviates this issue by utilizing a propagation scheme based on personalized PageRank. Their approach is able to achieve state-of-the-art classification performance by balancing the preservation of information from the local neighborhood and from the extended (multi-hop) neighborhood of a given node. However, since they explicitly perform a variant of power iteration during both training and inference

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MLG '19, August 05, 2019, Anchorage, AK
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

this approach does not easily scale to large graphs. The method we propose here adapts the propagation scheme of PPNP and scales to graphs with billions of nodes. Specifically, we utilize the (strong) localization properties [21, 39] of personalized PageRank vectors for real-world graphs which can be readily approximated with sparse vectors and efficiently pre-computed in a distributed manner [3]. Using these sparse pre-computed approximations we avoid explicitly performing the expensive power iteration procedure, while maintaining the influence of the relevant nodes multiple hops away.

2 RELATED WORK

2.1 (Scalable) Graph Neural Networks

Graph neural networks (GNNs) were first proposed in Gori et al. [22] and Scarselli et al. [44] and have since emerged as a powerful approach for solving many network mining tasks [7, 13, 14, 20, 23, 28, 32, 34, 44, 49]. Many of the proposed methods approach the problem by defining a notion of a graph convolution and can be roughly categorized into spectral- and spatial-based methods. Spectral-based graph convolutional networks (GCNs) [7, 14, 28, 32] utilize variants of the graph Laplacian matrix, which captures important properties of the graph and has been extensively studied in the fields of spectral graph theory [7] and graph signal processing [46]. On the other hand, spatial-based methods [13, 18, 20, 23, 37, 40, 44, 49] define the learnable filters in the domain of the nodes, where the convolution takes the form of an aggregation (often averaging) of the representations of a given central node and the representations of its neighbors.

Most graph neural networks do not scale well to large graphs since they typically need to perform a recursive neighborhood expansion to compute the hidden representations of a given node. While several approaches have been proposed to improve the efficiency of graph neural networks [10, 11, 18, 23, 25, 43, 55, 59], the scalability of GNNs to massive (web-scale) graphs is still understudied. Ying et al. [59] is the only work that demonstrates an approach that is applicable in practice, providing results on a graph with 3 billion nodes and 18 billion edges. The largest graphs considered by most of the scalable approaches are several orders of magnitude smaller (under 250K nodes).

Some scalability is achieved in Hamilton et al. [23] by sampling a *fixed* number of nodes from the k-hop neighborhood of a given node. Ying et al. [59] alleviate some limitations of this approach (e.g. needing to store the entire graph) by creating a sub-graph that contains only the given central nodes and their neighbors sampled according to an importance score. This score can be seen as an approximation of the standard (non-personalized) PageRank score, however the number of random walks required to achieve a good approximation is relatively high [16] making it a suboptimal choice in practice. Similarly, other scalable approaches [18, 43, 59] utilize a sampled sub-graph to tackle issues of limited (GPU) memory.

Gao et al. [18] collect the representations from a node's neighborhood into a matrix, sort independently along each column (each feature), and use the k largest entries as input to a 1-dimensional CNN. Chen et al. [10] directly sample the receptive field for each layer using importance sampling. They assume that the nodes of the input graph are i.i.d. samples of a possibly infinite graph under a given probability distribution. Chen et al. [11] use the historical activations of the nodes as a control variate and limit the receptive field to the 1-hop neighborhood. Huang et al. [25] propose an adaptive sampling strategy with a trainable sampler per layer. Sato et al. [43] propose a constant time approximation algorithm (independent of the number of nodes, edges, and neighbors) for inference and gradient computation of a GNN and give theoretical guarantees in terms of the approximation error. Nonetheless, all of these approaches rely on a multi-hop message-passing procedure which limits their scalability.

Buchnik and Cohen [8] propose (normalized Laplacian) feature propagation which can be viewed as a simplified linearized GNN. They perform graph-based smoothing as a preprocessing step (before learning) to obtain diffused node features which are then used to learn a simple logistic regression classifier to predict the node labels. Wu et al. [55] propose an equivalent simple graph convolution (SGC) model and show that it scales to large graphs while achieving performance comparable to less scalable state-of-the-art GNNs. Specifically, the node features are diffused by multiplication with the k-th power of the normalized adjacency matrix. However, often the node features are high dimensional making the preprocessing step computationally expensive. More importantly, while the node features are typically sparse the obtained diffused features become denser, which significantly reduces the efficiency of the subsequent learning step. Both of these approaches are a special case of the PPNP model [30] which experimentally shows higher classification performance [15, 30].

2.2 (Approximate) Personalized PageRank

PageRank and its many variants [1, 26, 52] have been extensively studied in the literature. In the context of this work we are interested in efficient and scalable algorithms for computing (an approximation) of personalized PageRank, and given the broad applicability of PageRank many such algorithms have been developed. Random walk sampling [16] is a common technique for approximating (personalized) PageRank. While this method is simple to implement, unfortunately in order to guarantee at most ϵ absolute error with probability of $1 - 1/n$ the number of random walks required is $O(\frac{\log n}{\epsilon^2})$. Forward search [3] and backward search [2] can be viewed as deterministic variants of the random walk sampling method where given a starting configuration the PageRank scores are updated by traversing the out-links (respectively, in-links) of the nodes. The approach based on Gauss-Southwell used in Gleich et al. [21] can also be considered as a variant of forward search.

More recent approaches [35, 50, 51] combine these basic techniques to create algorithms with enhanced guarantees. For example Wei et al. [54] propose the TopPPR algorithm combining the strengths of random walks, forward search, and backward search simultaneously. They are able to compute the top k entries of a personalized PageRank vector up to a user specified precision using a filter-refinement paradigm. Another family of approaches [17] are based on the idea of maintaining upper and lower bounds on the PageRank scores which are then used for early termination with certain guarantees. For this work we adapt the approach by Andersen et al. [3] since it offers a good balance of scalability, approximation guarantees and ease of (distributed) implementation.

3 MODEL

3.1 GNNs and Message-Passing

Many proposed GNN models can be analyzed using the message-passing framework proposed by Gilmer et al. [20] or other similar frameworks [4, 56]. Typically, the computation is carried out in two phases: (i) messages are propagated along the neighbors; and (ii) the messages are aggregated to obtain the updated representations. At each layer transformation of the input (e.g. linear projection plus a non-linearity) is coupled with aggregation/propagation among the neighbors (e.g. averaging). Crucially, computing the hidden representation for a given node requires considering its neighbors, and the neighbors in turn have to consider *their own* neighbors and so on. This process leads to a recursive neighborhood expansion growing with each additional layer.

Yet increasing the number of layers is desirable since: (i) it allows the model to incorporate information from more distant neighbors; and (ii) enables hierarchical feature extraction and thus learning richer node representations. However, this has both computational and modelling consequences. First, the recursive neighborhood expansion at each layer coupled with the small diameter [53] of real-world graphs means that to produce the output at the final layer for a single node we have to propagate messages on almost the entire graph. For large graphs necessarily stored (shared) on multiple machines gathering the required neighboring information requires many expensive remote procedure calls (RPCs). Second, it has been shown [33, 58] that naively stacking multiple layers may lead to over-smoothing, which results in poor predictive performance.

To tackle both of these challenges Klicpera et al. [30] suggest to decouple the feature transformation from the propagation. Predictions are first generated (e.g. with a neural network) for each node utilizing only that node's own features, and then propagated using an adaptation of personalized PageRank. Specifically, the output of their PPNP model is defined as:

$$Z = \text{softmax}(\mathbf{\Pi}^{\text{sym}} H), \qquad H_{i,:} = f_\theta(\boldsymbol{x}_i) \tag{1}$$

where $\mathbf{\Pi}^{\text{sym}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\tilde{\boldsymbol{A}})^{-1}$ is a symmetric propagation matrix, $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ is the normalized adjacency matrix with added self-loops, α is a teleport (restart) probability, $\boldsymbol{H}$ collects the individual per-node predictions for all nodes, and $\boldsymbol{Z}$ collects the final predictions after propagation. The local per-node predictions $\boldsymbol{H}_{i,:}$ are generated by a neural f_θ that processes the features $\boldsymbol{x}_i$ of every node i independently. The responsibility for learning good representations is delegated to f_θ, while $\mathbf{\Pi}^{\text{sym}}$ ensures that the representations are smoothly changing w.r.t. the graph.

Because directly calculating the dense propagation matrix $\mathbf{\Pi}^{\text{sym}}$ in Eq. 1 is inefficient, the authors propose to use a variant of power iteration to compute the final predictions instead. Unfortunately, even a moderate number of power iteration evaluations (e.g. Klicpera et al. [30] used $K = 10$ to achieve a good approximation) is prohibitively expensive for large graphs. Moreover, despite the fact that $\tilde{\boldsymbol{A}}$ is sparse, graphs beyond a certain size cannot be stored in memory.

3.2 Personalized PageRank and Localization

In contrast to PPNP our model uses the personalized PageRank matrix $\mathbf{\Pi}^{\text{ppr}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A})^{-1}$ to propagate information since it is more amenable to efficient approximation [2, 3, 16, 17, 21, 35, 50, 51, 54]. Here each row $\boldsymbol{\pi}(i) := \mathbf{\Pi}^{\text{ppr}}_{i,:}$ is equal to the personalized (seeded) PageRank vector of node i. When the network is strongly connected, $\boldsymbol{\pi}(i)$ is non-zero for all nodes. Nevertheless, we can obtain a good approximation of $\boldsymbol{\pi}(i)$ by truncating small elements to zero because of its (strong) localization behavior [3, 21, 39].

Gleich et al. [21] obtain a sparse approximation with accuracy guarantees using the Gauss-Southwell coordinate relaxation method. Andersen et al. [3] show that $\boldsymbol{\pi}(i)$ can be weakly approximated with a low number of non-zero entries using a scalable algorithm that applies a series of push operations which can be executed in a distributed manner. Intuitively, the above results indicate that most of the probability mass in the personalized PageRank vectors $\boldsymbol{\pi}(i)$ is localized on a small number of nodes, and that we can approximate $\boldsymbol{\pi}(i)$ with a sparse vector, which in turn means that we can approximate $\mathbf{\Pi}^{\text{ppr}}$ with a sparse matrix.

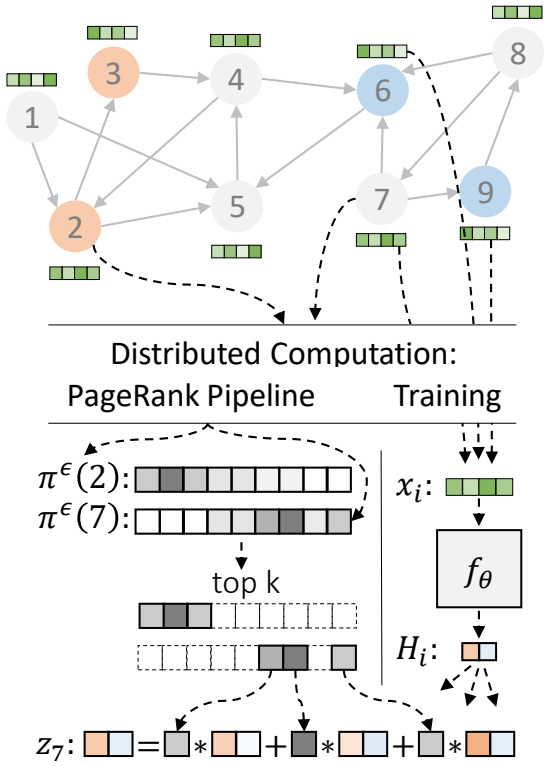

Figure 1: An illustration of our model PPRGo. For each node i first we pre-compute an approximation of its personalized PageRank vector $\boldsymbol{\pi}^{(\epsilon)}(i)$. The approximation is computed efficiently and in parallel using a distributed batch data processing pipeline. The final prediction z_i is then generated as a weighed average of the local (per-node) predictions $H_{j,:} = f_\theta(\boldsymbol{x}_j)$ for the top k nodes with largest personalized PageRank score $\boldsymbol{\pi}(i)_j$. To train the model $f_\theta(\cdot)$ that maps node attributes $\boldsymbol{x}_i$ to local predictions H_i we only need the personalized PageRank vectors of the training nodes and attributes of the their respective top k nodes. The model is trained in a distributed manner on multiple batches of data in parallel.

3.3 The PPRGo Model

The definition of our model is motivated by: (i) the insights from Sec. 3.1, namely that we can decouple the feature transformation from the information propagation, and (ii) the insights from Sec. 3.2, namely that we can approximate $\mathbf{\Pi}^{\text{ppr}}$ with a sparse matrix. Analogous to Eq. 1 we define the final predictions of the model (see overview of the model in Fig. 1).:

$$\boldsymbol{Z} = \text{softmax}\big(\mathbf{\Pi}^{(\epsilon)}\boldsymbol{H}\big), \qquad \boldsymbol{H}_{i,:} = f_\theta(\boldsymbol{x}_i) \tag{2}$$

where $\mathbf{\Pi}^{(\epsilon)}$ is a sparse approximation of $\mathbf{\Pi}^{\text{ppr}}$, which we obtain by adapting the push-flow algorithm described in Andersen et al. [3]. To enable further scalability we truncate $\mathbf{\Pi}^{(\epsilon)}$ to contain only the top k largest entries for each row. That is, for each node i we only consider the set of nodes with top k largest scores according to $\boldsymbol{\pi}(i)$. Focusing on the predictions for a given node i we have:

$$\boldsymbol{z}_i = \text{softmax}\Bigg(\sum_{j \in \mathcal{N}^k(i)} \boldsymbol{\pi}^{(\epsilon)}(i)_j \boldsymbol{H}_j\Bigg) \tag{3}$$

where $\mathcal{N}^k(i)$ enumerates the indices of the top k largest non-zero entries in $\boldsymbol{\pi}^{(\epsilon)}(i)$. Eq. 3 highlights that we only have to consider a small number of other nodes to compute the final prediction for a given node. Furthermore, this definition allows us to explicitly trade-off scalability and performance by increasing/decreasing the number of neighbors k we take into account. We can achieve a similar trade-off by increasing/decreasing the threshold ϵ.

In contrast to the PPNP model, a big advantage of PPRGo is that we can pre-compute the sparse matrix $\mathbf{\Pi}^{(\epsilon)}$ once before we start training. During training and inference we can then compute the predictions in $O(k)$ time, where $k \ll N$, and N is number of nodes. Better still, since for training we only require the rows of $\mathbf{\Pi}^{(\epsilon)}$ corresponding to the training nodes we can delay the computation of remaining rows, and furthermore, we only need to compute the predictions $f_\theta(\boldsymbol{x}_i)$ of the training nodes' top-k neighbors. More importantly, our model lends itself nicely to batched computation. E.g. for a batch of nodes of size b we have to load in memory the features of at most $b \cdot k$ nodes. In practice, this number could also be smaller than $b \cdot k$ since the nodes that appear in $\mathcal{N}^k(i)$ overlap for the different nodes in the batch.

To compute the approximations of the personalized PageRank vectors $\boldsymbol{\pi}^{(\epsilon)}$ we adapt the method proposed by Andersen et al. [3]. Instead of carrying out push-flow iterations until convergence, we perform a fixed number of iterations, and drop nodes whose residual score is below a specified threshold ϵ in each iteration. Additionally, we truncate nodes with a very large degree (≥ 10000) by randomly sampling their neighbors. The above modifications proved to be just as effective as Andersen et al. [3]'s method while being significantly faster in terms of runtime.

3.4 Effective Neighborhood, α and k

From the definition of the personalized PageRank matrix $\mathbf{\Pi}^{\text{ppr}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A})^{-1}$ we can observe that the hyper-parameter α controls the amount of information we are incorporating from the neighborhood of a node. Namely, for values of α close to 1 the random walks return (teleport) to the node i more often and we are therefore placing more importance on the immediate neighborhood of the node. As the value of α decreases to 0 we instead give more and more importance to the extended (multi-hop) neighborhood of the node. Intuitively, the importance of the k-hop neighborhood is proportional to $(1-\alpha)^k$. Note that the importance that each node assigns to itself (i.e. the value of $\boldsymbol{\pi}(i)_i$) is typically higher than the importance it assigns to the rest of the nodes.

In contrast to the message-passing framework, where in order to incorporate information from the extended neighborhood we have to add additional layers – thereby significantly increasing the computational complexity – in our model we can simply modify the teleport probability α. In conjunction with α, we can modify the number of k largest entries we consider to increase or decrease the size of the effective neighborhood. Alternatively, instead of considering the top k nodes for a fixed value of k we could adaptively choose k for every node i such that a given fraction of the probability mass from $\pi(i)$ (e.g. 90%) is accounted for. However, this would prevent us from batching the computation for several nodes using efficient matrix multiplication routines. Therefore, we use a fixed value of k since it has computational benefits while maintaining similar predictive performance to adaptive k.

3.5 Distributed Training

In contrast to most previously proposed methods [23, 55, 59] we utilize distributed computing techniques which significantly reduces the overall runtime of our method. Our model is trained in two stages: First, we pre-compute the approximated personalized PageRank vectors, and second, we train the model parameters with stochastic gradient descent. Both stages are implemented in a distributed fashion. For the first stage we use an efficient batch data processing pipeline [31] similar to MapReduce. Since we can compute the PageRank vectors for every node in parallel our implementation effortlessly scales to graphs with billions of nodes. Moreover, we can a priori determine the number of iterations we need for achieving a desired approximation accuracy [3, 21] which in term means we can reliably estimate the runtime beforehand.

We implement PPRGo in Tensorflow and optimize the parameters with distributed stochastic gradient descent. Specifically, the model parameters are stored on a parameter server (or several parameter servers depending on the model size) and multiple workers process the data in parallel. We use asynchronous training to avoid the communication overhead between many workers. Each worker fetches the most up-to-date parameters and computes the gradients for a mini-batch of data independently of the other workers.

3.6 Homophily and Model Limitations

Recall that we compute the representation of a given node as a weighted average of the representations of the nodes in its neighborhood where the weights are non-negative and (mostly) decreasing as we move away from the central node. This implies that the (final) learned representations for a node and its neighbors will be similar. This property of our model can be helpful or detrimental depending on whether the network exhibits homophily. The homophily principle [36] – birds of a feather flock together – in the context of node classification on graphs can be understood as the tendency of nodes that belong to the same class to form edges. That is, the higher the homophily in the network, the higher the ratio of edges

between nodes of the same class. Therefore, in networks with high homophily the learned representations will be tend to be similar for nodes that belong to the same class making the classification problem easier.

Alternatively, in networks with heterophily nodes between different classes tend to form edges. In this case, increasing the similarity between the representations of a node and its neighbors makes the classification problem more difficult. Fortunately, in most graphs the edges are indeed formed based on the principle of homophily. The homophily assumption is a limitation of our model. However, pragmatically one could simply check whether this assumption holds by estimating the rate of homophily using the labeled nodes.

4 EXPERIMENTS

Since this is a work-in-progress paper we present some initial experiments that provide a proof-of-concept, validate the proposed approach, and prompt us to further study PPRGo in the future.

4.1 Datasets

Most previously proposed approaches that focus on scalability and tackle the semi-supervised node classification task are evaluated on a small set of publicly available benchmark datasets [10, 11, 18, 23, 25, 43, 55]. The size of these common datasets in terms of number of nodes and edges in the graph is relatively small, with the Reddit graph (233K nodes, 11.6M edges) [23] being the largest.

In addition to the common benchmark datasets Wu et al. [55] also evaluate their approach on two other larger graphs, namely the Twitter user geolocation datasets: Twitter-US (0.45M nodes, 37.45M edges, 0.26M node features) [42], and Twitter-World (1.39M nodes, 3.39M edges, 0.12M node features) [24, 41]. The task is to infer the location of Twitter users based on the content of their Tweets and the information from their neighbors in a semi-supervised fashion given the ground-truth locations for some of the users. The graphs were extracted by Rahimi et al. [41] such that two users are connected if one mentions the other, or they co-mention another user. The node features for each user are a bag-of-word representation of the text in their tweets. Investigating the Twitter-World dataset we notice that around 70% of the nodes in the graph have only a self-loop and no other edges towards any other nodes. This means that this dataset does not have any meaningful graph structure and its usefulness for evaluating the performance of GNNs is limited. Therefore, we evaluate our approach only on Twitter-US.

To facilitate the development of scalable graph neural networks methods we create two new datasets. The first dataset is a citation graph based on the Open Academic Graph (OAG) where the nodes are papers and the edges denote references between them. The node features correspond to a bag-of-words representation of their abstracts. The OAG is generated by linking two large academic graphs: Microsoft Academic Graph (MAG) [47] and AMiner [48]. Similarly, we create the second dataset which is a co-authorship graph. We augmented both of these graph with "ground-truth" node labels corresponding to the field of study of the papers/authors respectively. We extract the node labels semi-automatically by first mapping the publishing venues (conferences and journals) to a field of study. The resulting graphs are few orders of magnitude larger then the commonly used benchmark graphs. We will release these datasets, including detailed description of the graph construction and node labelling process, as well as performance evaluations of PPRGo and other baselines in an updated version of the paper.

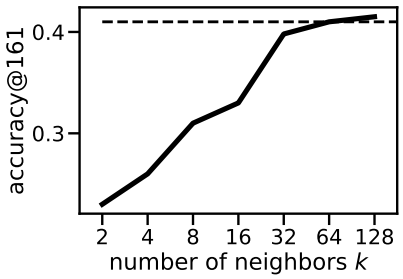

Figure 2: The average change in accuracy@161 on the Twitter-US dataset as we increase the number of neighbors (top k largest entries in $\boldsymbol{\pi}^{(\epsilon)}$) we use to compute Eq.3. The dotted line shows the performance of the previous SOTA [41].

4.2 Scalability vs. Performance Trade-off

The number of top-k nodes we are considering when aggregating the individual predictions for each node is an important hyperparameter (see Eq.3). To examine its effect on the performance of PPRGo we train our model on the Twitter-US graph for different values of k. We set the value of the teleport parameter $\alpha = 0.25$, and the approximation threshold $\epsilon = 10^{-5}$. We randomly sample 1% of the total number of nodes for training and additional 1% of nodes for validation (in contrast to e.g. Wu et al. [55] which use 95.5% of nodes for training and 2.2% for validation) since the sparsely labelled scenario is more relevant in practice. We repeat the experiment three times and report the average performance. As in previous works [41, 55] we evaluate the accuracy@161 which shows the accuracy of predicting the location of a user within 161km (or 100 miles) from the ground-truth location.

As we can see on Figure 2 the performance increases with k but starts to plateau at around top-$k = 64$ neighbors and reaches the previous state-of-the-art performance [41] indicated with the dashed line. The reason for this behavior becomes more clear by examining Figure 3. For each node i we calculate the sum of the top-k largest scores in $\boldsymbol{\pi}^{(\epsilon)}(i)$ and we plot the average and the standard deviation across all nodes. We see that by looking at very few nodes – e.g. 64 out of 0.45 million – we are able to capture the majority of the personalized PageRank scores on average (recall that $\sum_j \boldsymbol{\pi}^{(\epsilon)}(i)_j \leq 1$). Interestingly, the curves in both Figure 2 and Figure 3 plateau around the same value of k. These figures validate our approach of approximating the dense personalized PageRank vectors with their respective sparse top-k versions.

4.3 One-hop vs. Multi-hop

We aim to compare the performance of one-hop propagation using personalized PageRank and the traditional multi-hop message passing propagation. To make sure that we observe differences that are mostly due to the type of propagation and not other factors we implement a simple 2-hop GNN [28] which is also trained in a distributed manner using the same framework as PPRGo.

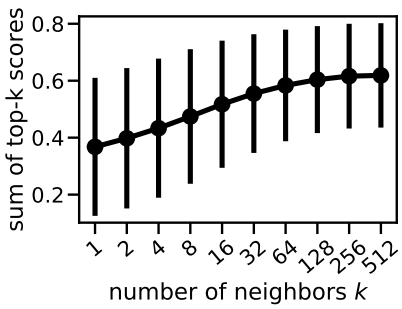

Figure 3: For each node i we calculate the sum of the top-k largest scores in $\pi^{(\epsilon)}(i)$ and we plot the average and the standard deviation across all nodes.

Table 1 shows that our method PPRGo which performs propagation only once at the end using personalized PageRank can achieve the same performance as expensive multi-hop message passing methods. Specifically, we show the relative accuracy – i.e. the accuracy of the best performing method is set to 100% and the rest of the results are calculated relative to it. We show relative speed improvement since the wall clock time depends on the number of worker machines used for distributed training. Here we make sure that both the 2-hop model and PPRGo are looking at the same number of neighbors, i.e. if PPRGo uses top-$k = 64$ then the 2-hop model uses information from $8 \times 8 = 64$ nodes from its first and second hop respectively. PPRGo is several orders of magnitude faster than the 2-hop model while maintaining competitive performance.

Table 1: Relative performance in terms of runtime and accuracy for a PPRGo and an equivalent 2-hop model using distributed training.

	Relative accuracy	Speed improvement
2-hop model	100%	–
PPRGo	97%	10^3

To further highlight the benefit of PPRGo we compare the runtime and performance of several methods using a single machine. Specifically, we run experiments on Nvidia 1080Ti GPUs using CUDA and TensorFlow and on Intel CPUs (20 cores). For SGC [55] we use the second power of the graph Laplacian (i.e. we have a 2-hop model). For GCN+H [41] we copy the result from the original paper. We compute the overall runtime including the time needed to pre-compute the propagation matrices for both PPRGo and SGC. As we can see in Table 2, PPRGo is able to significantly reduce the overall runtime while maintaining competitive accuracy.

Table 2: Relative performance in terms of runtime and accuracy on a single machine.

	accuracy@161	Runtime
GCN+H [41]	41.0	–
SGC [55]	14.3	2h 03m
PPRGo	41.5	0h 47m

5 CONCLUSION

In this work-in-progress paper we propose a graph neural network for semi-supervised node classification that can scale effortlessly to graphs with billions of nodes. In comparison to previous approaches our model does not rely on an expensive message-passing procedure. Our formulation allows to explicitly trade-off scalability and performance via a few intuitive hyper-parameters. Beyond the initial experiments which validate the proposed approach we aim to perform a thorough empirical evaluation in future work, including evaluation on the two proposed large scale benchmark datasets.

REFERENCES

[1] 1998. The PageRank Citation Ranking: Bringing Order to the Web.
[2] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. 2008. Local Computation of PageRank Contributions. *Internet Mathematics* 5 (2008), 23–45.
[3] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *null*. IEEE, 475–486.
[4] 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. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).
[5] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. (2018).
[6] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.
[7] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. [n.d.]. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* ([n. d.]).
[8] Eliav Buchnik and Edith Cohen. 2018. Bootstrapped graph diffusions: Exposing the power of nonlinearity. In *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*. ACM, 8–10.
[9] Deepayan Chakrabarti and Christos Faloutsos. 2006. Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)* 38, 1 (2006), 2.
[10] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).
[11] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction.. In *ICML*. 941–949.
[12] Zhengdao Chen, Lisha Li, and Joan Bruna. 2018. Supervised Community Detection with Line Graph Neural Networks. (2018).
[13] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. 2018. Learning Steady-States of Iterative Algorithms over Graphs. In *International Conference on Machine Learning*. 1114–1122.
[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.
[15] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR* abs/1903.02428 (2019). arXiv:1903.02428 http://arxiv.org/abs/1903.02428
[16] Dániel Fogaras and Balázs Rácz. 2004. Towards Scaling Fully Personalized PageRank. In *WAW*.
[17] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. 2013. Fast and Exact Top-k Algorithm for PageRank. In *AAAI*.
[18] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1416–1424.
[19] Lise Getoor and Christopher P Diehl. 2005. Link mining: a survey. *Acm Sigkdd Explorations Newsletter* 7, 2 (2005), 3–12.
[20] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212* (2017).
[21] David F Gleich, Kyle Kloster, and Huda Nassar. 2015. Localization in Seeded PageRank. *arXiv preprint arXiv:1509.00016* (2015).
[22] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, Vol. 2. IEEE, 729–734.
[23] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.
[24] Bo Han, Paul Cook, and Timothy Baldwin. 2012. Geolocation prediction in social media data by finding location indicative words. *Proceedings of COLING 2012*

(2012), 1045–1062.
[25] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. In *Advances in Neural Information Processing Systems*. 4563–4572.
[26] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *WWW*.
[27] Tatsuro Kawamoto, Masashi Tsubaki, and Tomoyuki Obuchi. 2018. Mean-field theory of graph neural networks in graph partitioning. In *NeurIPS*.
[28] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[29] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[30] Johannes Klicpera, Aleksandar Bojchevski, and Stephan GÃijnnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1gL-2A9Ym
[31] SPT Krishnan and Jose L Ugia Gonzalez. 2015. Google cloud dataflow. In *Building Your Next Big Thing with Google Cloud Platform*. Springer, 255–275.
[32] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. 2017. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv preprint arXiv:1705.07664* (2017).
[33] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. *arXiv preprint arXiv:1801.07606* (2018).
[34] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).
[35] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *WSDM*.
[36] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* 27, 1 (2001), 415–444.
[37] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. CVPR*, Vol. 1. 3.
[38] Federico Monti, Michael M. Bronstein, and Xavier Bresson. 2017. Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. In *NIPS*.
[39] Huda Nassar, Kyle Kloster, and David F Gleich. 2015. Strong localization in personalized PageRank vectors. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 190–202.
[40] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. 2014–2023.
[41] Afshin Rahimi, Trevor Cohn, and Tim Baldwin. 2018. Semi-supervised user geolocation via graph convolutional networks. *arXiv preprint arXiv:1804.08049* (2018).
[42] Stephen Roller, Michael Speriosu, Sarat Rallapalli, Benjamin Wing, and Jason Baldridge. 2012. Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 1500–1510.
[43] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. 2019. Constant Time Graph Neural Networks. *arXiv preprint arXiv:1901.07868* (2019).
[44] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.
[45] Uri Shaham, Kelly Stanton, Henry Li, Boaz Nadler, Ronen Basri, and Yuval Kluger. 2018. SpectralNet: Spectral Clustering using Deep Neural Networks. *arXiv preprint arXiv:1801.01587* (2018).
[46] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* 30, 3 (2013), 83–98.
[47] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In *WWW 2015*.
[48] Jie Tang, Jing Zhang, Limin Yao, Juan-Zi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: extraction and mining of academic social networks. In *KDD*.
[49] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* 1, 2 (2017).
[50] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: Effective Indexing for Approximate Personalized PageRank. *PVLDB* 10 (2016), 205–216.
[51] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD*.
[52] Xuanhui Wang, Azadeh Shakery, and Tao Tao. 2005. Dirichlet PageRank. In *SIGIR*.
[53] Duncan J. Watts. 1999. Networks , Dynamics , and the Small-World Phenomenon.
[54] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD Conference*.
[55] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying Graph Convolutional Networks. *arXiv preprint arXiv:1902.07153* (2019).
[56] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).
[57] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks? *arXiv preprint arXiv:1810.00826* (2018).
[58] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. *arXiv preprint arXiv:1806.03536* (2018).
[59] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *arXiv preprint arXiv:1806.01973* (2018).
[60] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. *arXiv preprint arXiv:1802.09691* (2018).
[61] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep Learning on Graphs: A Survey. *CoRR* abs/1812.04202 (2018).
[62] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *arXiv preprint arXiv:1812.08434* (2018).