Decoupling the Depth and Scope of Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

State-of-the-art Graph Neural Networks (GNNs) have limited scalability with respect to the graph and model sizes. On large graphs, increasing the model depth often means exponential expansion of the receptive field (*i.e.*, scope). Beyond just a few layers, two fundamental challenges emerge: 1. degraded *expressivity* due to oversmoothing, and 2. expensive *computation* due to neighborhood explosion. We propose a design principle to decouple the depth and scope of GNNs – to generate representation of a target entity (*i.e.*, a node or an edge), we first extract a localized subgraph as the *bounded-size* receptive field, and then apply a GNN of arbitrary depth on top of the subgraph. A properly extracted subgraph consists of a small number of critical neighbors, while excluding irrelevant ones. The GNN, no matter how deep it is, smooths the local signals into informative representation without oversmoothing the global graph into "white noise". Theoretically, such decoupling improves the expressive power of popular GNN architectures, including GCN, GraphSAGE and GIN. Empirically, on seven graphs (up to 110M nodes) and six backbone GNN architectures, our design achieves state-of-the-art accuracy with an order of magnitude reduction in computation and hardware cost.

1 Introduction

Graph Neural Networks (GNNs) have now become the state-of-the-art models for graph mining [48, 11, 56], facilitating applications such as social recommendation [34, 51, 36], knowledge understanding [39, 37, 57] and drug discovery [43, 31]. With the numerous architectures proposed [21, 10, 45, 49], it still remains an open question how to effectively scale up GNNs with respect to both the model size and graph size. There are two fundamental obstacles when we increase the number of GNN layers:

- *Expressivity* challenge (*i.e.*, oversmoothing [29, 35, 38, 16]): iterative mixing of neighbor features collapses embedding vectors of different nodes into a fixed low-dimensional subspace.
- *Computation* challenge (*i.e.*, neighbor explosion [5, 6, 7, 53]): recursive expansion of multi-hop neighborhood results in exponentially growing receptive field size.

To address the expressivity challenge, most remedies focus on neural architecture exploration: [45, 10, 49, 28] propose more expressive aggregation functions when propagating neighbor features. [50, 27, 17, 33, 1, 32, 30] impose residue-style design components to construct flexible and dynamic receptive fields. Among them, [50, 27, 17] use skip-connection across multiple GNN layers, and [33, 1, 32, 30] encourage multi-hop message passing within each single layer. As for the computation challenge, sampling methods have been explored to improve the training speed and efficiency. Importance based layer-wise sampling [6, 5, 59] and subgraph-based sampling [7, 53] alleviate neighbor explosion, while preserving training accuracy. Unfortunately, such sampling methods cannot be naturally generalized to inference without accuracy loss (see also Section 4).

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

The above lines of research have only guided us to partial solutions. Yet what is the root cause of both the expressivity and computation challenges? Setting aside the design of GNN architectures or sampling schemes, we provide an alternative perspective by interpretting the *data* in a different way.

Two views on the graph. Suppose we are given an input graph $\mathcal{G}$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$. The most straightforward way is to view $\mathcal{G}$ as a single *global* graph. So any two nodes u and v belong to the same $\mathcal{G}$, and if u and v lie in one connected component, they will ultimately see each other in their own neighborhood no matter how far away u and v are. Alternatively, we can take a *local* view. For each node v, there is a *latent* $\mathcal{G}_{[v]}$ surrounding it which captures the characteristics of just v. The full $\mathcal{G}$ is resulted from the *union* of all such $\mathcal{G}_{[v]}$. Thus, $\mathcal{V}_{[v]}$ rather than $\mathcal{V}$ defines v's neighborhood: if $u \notin \mathcal{V}_{[v]}$, v will never consider u as a neighbor. Our design is based on the local view.

Receptive field of GNNs. Recall the expressivity and computation challenges. Both are closely related to the enlargement of the GNN receptive field. On the other hand, how we define the receptive field is directly related to how we view $\mathcal{G}$. With a global view, an L-layer GNN has the receptive field of the full L-hop neighborhood. With a local view, the receptive field is simply $\mathcal{V}_{[v]}$ regardless of the GNN depth. The two lines of research mentioned above, one on architectural exploration and the other on sampling, both take the *global* view.[1] Consequently, the depth and receptive field of the GNN are *tightly coupled*. Such coupling significantly limits the design space exploration of GNNs with various depths [52]. Consider the example of `ogbn-products`, a medium-scale graph in Open Graph Benchmark [14]. The average number of 4-hop neighbors is around 0.6M, corresponding to 25% of the full graph size. For such a GNN to generate representation of a single target node, it needs to process features in this huge 0.6M-large receptive field. Since most of the 0.6M nodes would be barely relevant to the target node, propagating their features is inefficient, useless and even harmful.

Present work: Decoupling the depth and receptive field. By taking a local view on $\mathcal{G}$, we propose a general design principle to decouple the depth and receptive field (scope) of the GNN. To generate a node representation, we first extract a small subgraph of $\mathcal{G}$ surrounding the target node (*i.e.*, $\mathcal{G}_{[v]}$). On top of the subgraph, we apply a GNN whose number of layers and message passing functions can be flexibly chosen. The decoupling enables us to customize the receptive field and tune the number of layers independently. Such a design principle then leads to the a practical implementation, SHADOW-GNN: Decoupled GNN on a shallow subgraph. In SHADOW-GNN, the subgraph is shallow, since it only contains a fraction of the L-hop neighbors in $\mathcal{G}$, and in practice, L also only takes small values such as 2 or 3; The GNN has depth (number of layers) $L' \geq L$, since its depth is decoupled from its receptive field. A SHADOW-GNN addresses the expressivity and computation challenges simultaneously. Firstly, *why shallow subgraphs*? The most important reason is, for many real-world graphs, the relevance of the neighbors decreases rapidly with the distance to the target node. *e.g.*, in a social network, the friend of a friend of a friend may share little commonality with you. In addition, a shallow and small receptive field automatically guarantees efficient GNN training and inference. Secondly, *why decoupled GNN*? Using more layers (L') than hops (L) means the same pair of subgraph nodes may exchange messages multiple times. The extra message passing helps GNN better absorb and embed the subgraph information. Theoretically, we prove that SHADOW-GNN improves the expressive power for three backbone architectures: SHADOW-GCN does not oversmooth even with infinite depth; SHADOW-SAGE can learn a function where GraphSAGE cannot; SHADOW-GIN can be more powerful than the 1-dimensional Weisfeiler-Lehman test [40].

Empirically, we show our "decoupling" design improves both the accuracy and computation efficiency. On seven benchmarks (including the largest `ogbn-papers100M` graph with 110M nodes) and across two graph learning tasks, SHADOW-GNN achieves significant accuracy gains compared to the original models. Meanwhile, the computation and hardware costs are reduced by orders of magnitude.

2 Preliminaries

Let $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ be an undirected graph, with node set $\mathcal{V}$, edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and node feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$. Let $\mathcal{N}_v$ denote the set of v's direct neighbors in $\mathcal{G}$. The uth row of $\boldsymbol{X}$ corresponds to the length-d feature of node u. Let $\boldsymbol{A}$ be the adjacency matrix of $\mathcal{G}$ where $A_{u,v} = 1$ if edge $(u, v) \in \mathcal{E}$ and $A_{u,v} = 0$ otherwise. Let $\boldsymbol{D}$ be the diagonal degree matrix of $\boldsymbol{A}$. Denote $\widetilde{\boldsymbol{A}} = \boldsymbol{D}_*^{-\frac{1}{2}} \boldsymbol{A}_* \boldsymbol{D}_*^{-\frac{1}{2}}$ as the adjacency matrix after symmetric normalization ("$*$" means augmented

[1] The sampling methods [6, 10, 5, 59, 53, 8] aim at estimating the quantities of the full L-hop neighborhood

with self-edges), and $\widehat{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}$ as the one after random walk normalization. Let subscript "$[u]$" mark the quantities corresponding to a subgraph surrounding node u. For example, the subgraph itself is $\mathcal{G}_{[u]}$. Subgraph matrices $\boldsymbol{X}_{[v]}$ and $\boldsymbol{A}_{[v]}$ have the same dimension as the original $\boldsymbol{X}$ and $\boldsymbol{A}$. Yet, row vector $\left[\boldsymbol{X}_{[v]}\right]_u = \boldsymbol{0}$ for $u \notin \mathcal{V}_{[v]}$. Element $\left[\boldsymbol{A}_{[v]}\right]_{u,w} = 0$ if either $u \notin \mathcal{V}_v$ or $w \notin \mathcal{V}_{[v]}$. For an L-layer GNN, let superscript "(ℓ)" denote the layer-ℓ quantities. Let $d^{(\ell)}$ be the number of channels for layer ℓ; $\boldsymbol{H}^{(\ell-1)} \in \mathbb{R}^{|\mathcal{V}| \times d^{(\ell-1)}}$ and $\boldsymbol{H}^{(\ell)} \in \mathbb{R}^{|\mathcal{V}| \times d^{(\ell)}}$ be the input and output features. So $\boldsymbol{H}^{(0)} = \boldsymbol{X}$ and $d^{(0)} = d$. Further, let σ be the activation and $\boldsymbol{W}^{(\ell)}$ be the learnable weight. For example, a GCN layer performs $\boldsymbol{H}^{(\ell)} = \sigma\left(\widetilde{\boldsymbol{A}}\boldsymbol{H}^{(\ell-1)}\boldsymbol{W}^{(\ell)}\right)$. A GraphSAGE layer performs $\boldsymbol{H}^{(\ell)} = \sigma\left(\boldsymbol{H}^{(\ell-1)}\boldsymbol{W}_1^{(\ell)} + \widehat{\boldsymbol{A}}\boldsymbol{H}^{(\ell-1)}\boldsymbol{W}_2^{(\ell)}\right)$.

Our analysis in Section 3 mostly focuses on the node classification task. Yet our design can be generalized to the link prediction task, as demonstrated by our experiments in Section 5.

Definition 2.1. *(Depth of subgraph) Assume the subgraph $\mathcal{G}_{[v]}$ is connected. The depth of $\mathcal{G}_{[v]}$ is defined as $\max_{u \in \mathcal{V}_{[v]}} d(u, v)$, where $d(u, v)$ denotes the shortest path distance from u to v.*

3 Decoupling the Depth and Scope of GNNs

"Decoupling the depth and scope of GNNs" is a design principle to improve the expressivity and efficiency of GNNs without modifying the layer architecture. We implement such a principle by SHADOW-GNN. Compared with a normal GNN, SHADOW-GNN consists of an additional component: the subgraph extractor `EXTRACT`. To generate embedding of a target node v, SHADOW-GNN proceeds as follows: We first use `EXTRACT` $(v, \mathcal{G})$ to return a connected $\mathcal{G}_{[v]}$, where $\mathcal{G}_{[v]}$ is a subgraph containing v, and the depth of $\mathcal{G}_{[v]}$ is L. Then, we build an L'-layer GNN on $\mathcal{G}_{[v]}$ by treating $\mathcal{G}_{[v]}$ as the *new* full graph and by ignoring all nodes / edges not in $\mathcal{G}_{[v]}$. Effectively, $\mathcal{G}_{[v]}$ becomes the scope (*i.e.*, receptive field) of the GNN. The key point reflecting the "decoupling" is that $L' \geq L$.

A normal GNN is a special SHADOW-GNN. Under the normal setup, an L-layer GNN operates on the full $\mathcal{G}$ and propagates the influence from all the neighbors up to L hops away from v. Such a GNN is equivalent to a SHADOW-GNN when `EXTRACT` returns the full L-hop subgraph and $L' = L$.

The SHADOW-GNN design is motivated by the following:

1. A shallow neighborhood is *necessary* and *sufficeint* for the GNN to learn well.
2. Decoupling the depth and scope improves GNN *expressivity*.

The "sufficiency" of Point 1 can be seen in many real-world scenarios. For example, your close friends may be at most 2 hops away in the social network. More formally, the γ-decaying theorem [54] states that it only takes a shallow neighborhood to accurately estimate various important graph metrics. On the "necessity", note that the neighborhood size may grow exponentially with the distance to the target node. A deep neighborhood would be dominated by nodes irrelevant to the target node. The corresponding GNN would then need to differentiate the many useless nodes from the few useful ones, before it can extract meaningful features from the useful nodes.

Using three popular GNN architectures as representatives, we demonstrate how SHADOW-GNN improves expressivity from three different angles. On SHADOW-GCN (Section 3.1), we come from the *graph signal processing* perspective. The GCN propagation can be interpreted as applying filtering on the node signals [47]. Deep models correspond to high-pass filters. Applying such filtering on the local graph $\mathcal{G}_{[v]}$ preserves much richer information than applying on the global $\mathcal{G}$. On SHADOW-SAGE (Section 3.2), we view the GNN as a *function approximator*. We study how decoupling the GNN depth reduces the approximation error on a target function. On SHADOW-GIN (Section 3.3), we focus on learning *topological information*. We show that decoupling helps capture local graph structure which the powerful 1-dimensional Weisfeiler-Lehman test fails to capture.

3.1 Expressivity Analysis on SHADOW-GCN

GCNs [21] suffer from "oversmoothing" [29] – Each GCN layer smooths the features of the direct (*i.e.*, 1-hop) neighbors, and many GCN layers smooths the features of the full graph. Eventually,

such repeated smoothing process propagates to any target node just the averaged feature of all $\mathcal{V}$. "Oversmoothing" thus incurs significant information loss by wiping out all local information.

Formally, suppose the original features $\boldsymbol{X}$ reside in a high-dimensional space $\mathbb{R}^{|\mathcal{V}|\times d}$. Oversmoothing pushes $\boldsymbol{X}$ towards a low-dimensional subspace $\mathbb{R}^{|\mathcal{V}|\times d'}$, where $d' < d$. Corresponding analysis comes from two perspectives: oversmoothing by *a deep GCN*, and oversmoothing by *repeated GCN-style propagation*. The former considers the full neural network with non-linear activation, weight and bias. The later characterizes the aggregation matrix $\boldsymbol{M} = \lim_{L\to\infty} \widetilde{\boldsymbol{A}}^L \boldsymbol{X}$. It is shown that even with the vanilla architecture, a deep GCN with bias parameters does *not* oversmooth [16]. In addition, various tricks [58, 38, 33] can prevent oversmoothing from the neural network perspective. However, a deep GCN still suffers from accuracy drop, indicating that the GCN-style propagation (rather than other GCN components like activation and bias) may be the fundamental reason causing difficulty in learning. Therefore, we study the asymptotic behavior of the aggregation matrix $\boldsymbol{M}$ under the normal and SHADOW design. In other words, here in Section 3.1, we ignore the non-linear activation and bias parameters. Such setup is consistent with many existing literature such as [29, 32, 33, 58].

Proposition 3.1. *∞ number of feature propagation by* SHADOW-GCN *leads to*

$$\boldsymbol{m}_{[v]} = \left[e_{[v]}\right]_v \cdot \left(\boldsymbol{e}_{[v]}^{\mathsf{T}} \boldsymbol{X}_{[v]}\right) \tag{1}$$

where $\boldsymbol{e}_{[v]}$ is defined by $\left[e_{[v]}\right]_u = \sqrt{\frac{\delta_{[v]}(u)}{\sum_{w\in\mathcal{V}_{[v]}} \delta_{[v]}(w)}}$; $\delta_{[v]}(u)$ returns the degree of u in $\mathcal{G}_{[v]}$ plus 1.

Oversmoothing by normal GCN propagation. With a large enough L, the full L-hop neighborhood becomes $\mathcal{V}$ (assuming connected $\mathcal{G}$). So $\forall\ u, v$, we have $\mathcal{G}_{[u]} = \mathcal{G}_{[v]} = \mathcal{G}$, implying $\boldsymbol{e}_{[u]} = \boldsymbol{e}_{[v]}$ and $\boldsymbol{X}_{[u]} = \boldsymbol{X}_{[v]} = \boldsymbol{X}$. From Proposition 3.1, the aggregation converges to a point where *no* feature and *little* structural information of the target is preserved. The only information in $\boldsymbol{m}_{[v]}$ is v's degree.

Local-smoothing by SHADOW-GCN propagation. With a fixed subgraph, no matter how many times we aggregate using $\widetilde{\boldsymbol{A}}_{[v]}$, the layers will not include the faraway irrelevant nodes. From Proposition 3.1, $\boldsymbol{m}_{[v]}$ is a linear combination of the neighbor features $\boldsymbol{X}_{[v]}$. Increasing the number of layers only pushes the *coefficients* of each neighbor features to the stationary values. The *domain* $\boldsymbol{X}_{[v]}$ of such linear transformation is solely determined by `EXTRACT` and is independent of the model depth. Intuitively, if `EXTRACT` picks non-identical subgraphs for two nodes u and v, the aggregations should be different due to the different domains of the linear transformation. Therefore, SHADOW-GCN preserves *local* feature information whereas normal GCN preserves *none*. For structural information in $\boldsymbol{m}_{[v]}$, note that $\boldsymbol{e}_{[v]}$ is a normalized degree distribution of the subgraph around v, and $\left[e_{[v]}\right]_v$ indicates the role of the target node in the subgraph. By simply letting `EXTRACT` return the 1-hop subgraph, $\left[e_{[v]}\right]_v$ alone already contains all the information preserved by a normal GCN, which is v's degree in $\mathcal{G}$. Since $\boldsymbol{e}_{[v]}$ additionally reflects v's ego-net structure, a deep SHADOW-GCN with a naïve 1-hop `EXTRACT` preserves more structural information than a deep GCN.

Theorem 3.2. *Let $\overline{\boldsymbol{m}}_{[v]} = \phi_{\mathcal{G}}(v) \cdot \boldsymbol{m}_{[v]}$ where $\phi_{\mathcal{G}}$ is any non-zero function only depending on the structural property of v. Let $\mathcal{M} = \{\overline{\boldsymbol{m}}_{[v]} \mid v \in \mathcal{V}\}$. Given $\mathcal{G}$, `EXTRACT` and some continuous probability distribution in $\mathbb{R}^{|\mathcal{V}|\times d}$ to generate $\boldsymbol{X}$, then $\overline{\boldsymbol{m}}_{[v]} \neq \overline{\boldsymbol{m}}_{[u]}$ if $\mathcal{V}_{[u]} \neq \mathcal{V}_{[v]}$, almost surely.*

Corollary 3.2.1. *Consider `EXTRACT`$_1$, where $\forall v \in \mathcal{V},\ \left|\mathcal{V}_{[v]}\right| \leq n$. Then $|\mathcal{M}| \geq \left\lceil \frac{|\mathcal{V}|}{n} \right\rceil$ a.s.*

Corollary 3.2.2. *Consider `EXTRACT`$_2$, where $\forall\ u, v \in \mathcal{V},\ \mathcal{V}_{[v]} \neq \mathcal{V}_{[u]}$. Then $|\mathcal{M}| = |\mathcal{V}|$ a.s.*

Theorem 3.2 proves SHADOW-GCN does not oversmooth: 1. A normal GCN pushes the aggregation of same-degree nodes to the same point, while SHADOW-GCN with `EXTRACT`$_2$ ensures any two nodes (even with the same degree) have different aggregation. 2. A normal GCN wipes out all information in $\boldsymbol{X}$ after many times of aggregation, while SHADOW-GCN always preserves feature information. Particularly, with $\phi_{\mathcal{G}}(v) = \left(\delta_{[v]}(v)\right)^{-1/2}$, a normal GCN generates only one unique value of $\overline{\boldsymbol{m}}$ for all v, while SHADOW-GNN generates $|\mathcal{V}|$ different values for any $\phi_{\mathcal{G}}$ function.

3.2 Expressivity Analysis on SHADOW-SAGE

We compare the expressivity by showing 1. SHADOW-SAGE can express all functions GraphSAGE can, and 2. SHADOW-SAGE can express some functions GraphSAGE cannot. Recall, a GraphSAGE

layer performs the following: $\boldsymbol{h}_v^{(\ell)} = \sigma\left(\left(\boldsymbol{W}_1^{(\ell)}\right)^\mathsf{T}\boldsymbol{h}_v^{(\ell-1)} + \left(\boldsymbol{W}_2^{(\ell)}\right)^\mathsf{T}\left(\frac{1}{|\mathcal{N}_v|}\sum_{u\in\mathcal{N}_v}\boldsymbol{h}_u^{(\ell-1)}\right)\right)$. We can prove Point 1 by making an L'-layer SHADOW-SAGE identical to a L-layer GraphSAGE, by 1. letting `EXTRACT` return the full L-hop neighborhood, and 2. setting $\boldsymbol{W}_1^{(\ell)} = \boldsymbol{I}$, $\boldsymbol{W}_2^{(\ell)} = \boldsymbol{0}$ for $L+1 \le \ell \le L'$. For point 2, we consider a target function: $\tau\left(\boldsymbol{X}, \mathcal{G}_{[v]}\right) = \frac{1}{|\mathcal{V}_{[v]}|}\sum_{u\in\mathcal{V}_{[v]}}\boldsymbol{x}_u$ for some neighborhood $\mathcal{G}_{[v]}$. The feature mean of proper $\mathcal{G}_{[v]}$ contains rich information about v. Thus, an expressive model should learn τ well.

We conclude that GraphSAGE cannot learn τ accurately, while SHADOW-SAGE can approximate τ with error decaying exponentially with the model depth. To see this, let the depth of $\mathcal{G}_{[v]}$ be L. For GraphSAGE, we need to set its model depth $L' = L$. Otherwise, layers 1 to $L'-L$ will see nodes $v' \notin \mathcal{V}_{[v]}$. The only way to always exclude the impact of v' is by setting $\boldsymbol{W}_2 = \boldsymbol{0}$ for all such layers. Consequently, an L'-layer model becomes equivalent to an L-layer one. Next, by linearity of τ, `ReLU` is useless and should be bypassed by shifting $\boldsymbol{X}$ with bias. We can simplify further by setting feature dimension $d = 1$. Such GraphSAGE then outputs $\boldsymbol{\zeta} = \left[\sum_{\ell=1}^{L} w_\ell \widehat{\boldsymbol{A}}^\ell \boldsymbol{X}\right]_{v,:}$, with scalar parameters w_ℓ. To have $\boldsymbol{\zeta} = \boldsymbol{\tau}$, we need vector $\boldsymbol{z} = \left[\sum_{\ell=1}^{L} w_\ell \widehat{\boldsymbol{A}}^\ell\right]_{v,:}$ to have uniform values for all positions corresponding to $\mathcal{V}_{[v]}$. We can construct numerous depth-L examples on $\mathcal{G}_{[v]}$ (*e.g.*, see Appendix) such that $\boldsymbol{z}$ can never satisfy our condition. On the other hand, for SHADOW-SAGE, let `EXTRACT` return $\mathcal{G}_{[v]}$. Set $w_{L'} = 1$ and $w_\ell = 0$ for $1 \le \ell < L'$. Such a model outputs $\boldsymbol{\zeta}' = \left[\widehat{\boldsymbol{A}}_{[v]}^{L'}\boldsymbol{X}_{[v]}\right]_{v,:}$. When $L' \to \infty$, $\widehat{\boldsymbol{A}}_{[v]}^{L'}$ converges to $\frac{1}{|\mathcal{V}_{[v]}|}\boldsymbol{1}_{[v]}$ and thus $\boldsymbol{\zeta}'$ converges to τ, at an exponential rate [26]. The analysis on SHADOW-GNN holds for any connected $\mathcal{G}_{[v]}$ and any dimension d.

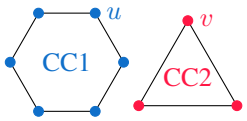
We have the following conclusions from the above analysis: 1. SHADOW-SAGE is more expressive than GraphSAGE. 2. appropriate `EXTRACT` function improves SHADOW-GNN expressivity, 3. In some cases, it may be desirable to set the SHADOW-GNN depth much larger than the subgraph depth.

3.3 Expressivity Analysis with More General GNN Backbones

While GCN and GraphSAGE are popular architectures in practice, they are not the theoretically most discriminative ones. It has been shown in [49] that discriminativeness of GNNs are upper-bound by the 1 dimensional Weisfeiler-Lehman test (*i.e.*, 1-WL) and GIN is an example architecture achieving the same discriminativeness as 1-WL. We show that applying the decoupling principle can further improve the discriminativeness of such GNNs, making them more powerful than 1-WL.

Recall that 1-WL is a graph isomorphism test aiming at distinguishing graphs of different structure. A GNN as expressive as 1-WL thus well captures the topological property of the target node. While 1-WL is already very powerful, it may still fail in some cases. *e.g.*, it cannot distinguish certain non-isomorphic, *regular* graphs. To understand why SHADOW-GNN works, we first need to understand why 1-WL fails. In a regular graph, all nodes have the same degree, and thus the "regular" property describes a *global* topological symmetry among nodes. Unfortunately, 1-WL (and the corresponding normal GNN) also operates *globally* on $\mathcal{G}$. So intuitively, on two different regular graphs, there is no way for 1-WL (and the normal GNN) to assign different labels by breaking such symmetry.

SHADOW-GNN can break such symmetry. Recall in Section 1 that SHADOW-GNN is built from the *local* perspective in the first place. In our example, *a subgraph of a regular graph may not be regular*. Thus, SHADOW-GNN can distinguish some nodes which a normal GNN cannot. We show this with the example in the left figure, where the graph consists of two connected components (CC). Both CC are 2-regular. Suppose node features are identical. Our goal is to discriminate u in CC1 from v in CC2 by their topological difference. No matter how many iterations 1-WL runs, or how many layers the normal GNN has, they cannot distinguish u and v. On the other hand, a SHADOW-GNN with 1-hop `EXTRACT` and at least 2 layers can discriminate u and v.



Theorem 3.3. *Consider GNNs whose layer function is defined by*

$$\boldsymbol{h}_v^{(\ell)} = f_1^{(\ell)}\left(\boldsymbol{h}_v^{(\ell-1)}, \sum_{u\in\mathcal{N}_v} f_2^{(\ell)}\left(\boldsymbol{h}_v^{(\ell-1)}, \boldsymbol{h}_u^{(\ell-1)}\right)\right), \tag{2}$$

where $f_1^{(\ell)}$ and $f_2^{(\ell)}$ are the update and message functions of layer-ℓ, implemented as MLPs. Then, such SHADOW-GNN *is more discriminative than the 1-dimensional Weisfeiler-Lehman test.*

See Appendix A for the proof. The theorem also implies that SHADOW-GIN is more discriminative than a normal GIN due to the correspondence between GIN and 1-WL.

3.4 Subgraph Extraction Algorithms

Our decoupling principle does not rely on specific subgraph extraction algorithms. Appropriate `EXTRACT` can be customized given the characteristic of $\mathcal{G}$, and different `EXTRACT` leads to different *implementation* of our decoupling principle. In general, there are three approaches to design `EXTRACT`: 1. *heuristic based*, where we pick graph metrics that reflect neighbor importance and then design `EXTRACT` by such metrics; 2. *model based*, where we assume a generation process on $\mathcal{G}$ and set `EXTRACT` as the reverse process, and 3. *learning based*, where we integrate the design of `EXTRACT` as part of the GNN training. In Section 5, we experiment with a few heuristic based `EXTRACT`, and thus we present some corresponding examples in the following. We leave detailed evaluation on the model and learning based `EXTRACT` as future work. See also Appendix for details.

Example heuristic based `EXTRACT`. The algorithm is derived from the selected graph metrics. For example, with the metric being shortest path distance, we design a L-hop extractor. *i.e.*, `EXTRACT` returns the full set or subset of the target node's L-hop neighbors in $\mathcal{G}$. Picking the random walk landing probability as the metric, we would a PPR extractor. *i.e.*, we first compute Personalized PageRank (PPR) on $\mathcal{G}$ to derive the PPR score of other nodes relative to the target node. Then `EXTRACT` define $\mathcal{V}_{[v]}$ by picking the top-K nodes with the highest PPR score. One can easily extend this approach by using other metrics such as Katz index [19], SimRank [18] and feature similarity.

3.5 Architecture

Subgraph pooling. In a normal GNN, the target receives messages from all neighbors, but two neighbor nodes may not have a chance to exchange any message to each other (*e.g.*, two nodes L-hop away from the target may be $2L$ hops from each other). Thanks to our decoupling principle, a deep SHADOW-GNN can pass messages multiple times between *any* pair of neighbors. Therefore, the final SHADOW-GNN layer outputs meaningful embeddings not only for the target, but also for any subgraph node. In the vanilla SHADOW-GNN, those neighbor embeddings are simply discarded. Alternatively, we can summarize them to further improve the target embedding.

Equivalently, decoupling connects the node- / link-level task with the graph-level task. *e.g.*, to classify a node is seen as to classify the subgraph around the node. Consequently, we can apply any subgraph pooling / READOUT operation (*e.g.*, [55, 25, 3]) at the end of the SHADOW-GNN propagation. In particular, "discarding all neighbor features" in the vanilla SHADOW-GNN design can be seen as performing the center pooling. See Appendix D and F.3 for algorithm and experiments.

Subgraph ensemble. It may be challenging in practice to design a single `EXTRACT` capturing all meaningful characteristics of the neighborhood. We can use multiple `EXTRACT` to jointly define the receptive field, and then ensemble multiple SHADOW-GNN at the subgraph level. Consider C candidates $\{\texttt{EXTRACT}_i\}$, each returning $\mathcal{G}^i_{[v]}$. To generate v's embedding, we first use C branches of L'-layer GNN to obtain intermediate embeddings for each $\mathcal{G}^i_v$, and then aggregate the C embeddings by some learnable function g. In practice, we design g as an attention based aggregation function (see Appendix). Subgraph ensemble is useful both when $\{\texttt{EXTRACT}_i\}$ consists of different algorithms and when each $\texttt{EXTRACT}_i$ performs the same algorithm under different parameters.

CASE STUDY Consider PPR-based $\texttt{EXTRACT}_i$ with different threshold θ_i on the neighbor PPR score. A SHADOW-GNN-ensemble can approximate PPRGo [4]. PPRGo generates embedding as: $\boldsymbol{\xi}_v = \sum_{u\in\mathcal{V}_{[v]}} \pi_u \boldsymbol{h}_v$, where π_u is u's PPR score and $\boldsymbol{h}_v = \texttt{MLP}(\boldsymbol{x}_v)$. We can partition $\mathcal{V}_{[v]} = \bigcup_{i=1}^C \mathcal{V}^i_{[v]}$ s.t. nodes in $\mathcal{V}^i_{[v]}$ have similar PPR scores denoted by $\widetilde{\pi}_i$, and $\widetilde{\pi}_i \le \widetilde{\pi}_{i+1}$. So $\boldsymbol{\xi}_v = \sum_{i=1}^C \rho_i \left(\sum_{u\in\mathcal{V}'_i} \boldsymbol{h}_u\right)$, where $\rho_i = \widetilde{\pi}_i - \sum_{j<i} \widetilde{\pi}_j$ and $\mathcal{V}'_i = \bigcup_{k=i}^C \mathcal{V}^k_{[v]}$. Now for each branch of SHADOW-GNN-ensemble, let parameter $\theta_i = \widetilde{\pi}_i$ so that $\texttt{EXTRACT}_i$ returns $\mathcal{V}'_i$. The GNN on $\mathcal{V}'_i$ can then learn $\sum_{u\in\mathcal{V}'_i} \boldsymbol{h}_u$ (see Section 3.2). Finally, set the ensemble weight as ρ_i. SHADOW-GNN-ensemble learns $\boldsymbol{\xi}_v$. As `EXTRACT` also preserves graph topology, our model can be more expressive than PPRGo.

4 Related Work

Deep GNNs. To improve the GNN performance while increasing the model depth, various layer architectures have been proposed. AS-GCN [17], DeepGCN [27], JK-net [50], MixHop [1], Snowball [32], DAGNN [30] and GCNII [33] all include some variants of residue connection, either across multiple layers or within a single layer. In principle, such architectures can also benefit the feature propagation of a deep SHADOW-GNN, since their design does not rely on a specific neighborhood (*e.g.*, L-hop). In addition to architectures, DropEdge [38] and Bayesian-GDC [12] propose regularization techniques by adapting dropout [42] to graphs. Such techniques are only applied during training, and inference may still suffer from issues such as oversmoothing.

Sampling based methods. Neighbor or subgraph sampling techniques have been proposed to improve training efficiency. FastGCN [6], VR-GCN [5], AS-GCN [17], LADIES [59] and MVS-GNN [8] sample neighbor nodes per GNN layer. Cluster-GCN [7] and GraphSAINT [53] sample a subgraph as the training minibatch. While sampling also changes the receptive field, all the above methods are fundamentally different from ours. The training samplers aim at estimating the quantities related to the full graph (*e.g.*, the aggregation of the full L-hop neighborhood), and so the inference model still operates on the full neighborhood to avoid accuracy loss. For SHADOW-GNN, since the decoupling principle is derived from a local view on $\mathcal{G}$, our `EXTRACT` does not estimate any full neighborhood quantities. Consequently, the sampling based methods only improve the training efficiency, while SHADOW-GNN addresses the computation challenge for both training and inference.

Re-defining the neighborhood. Various works reconstruct the original graph and apply the GNN on the re-defined neighborhood. GDC [23] views the reconstructed adjacency matrix as the diffusion matrix. SIGN [9] applies reconstruction for customized graph operators. AM-GCN [46] utilizes the reconstruction to separate feature and structure information. The above work re-define the neighborhood. However, they still have decoupled depth and receptive field. SHADOW-GNN can also work with the reconstructed graph $\mathcal{G}'$ by simply applying `EXTRACT` on $\mathcal{G}'$. In addition, in Appendix, we show how SHADOW-GNN on $\mathcal{G}$ can be reduced to a normal feature propagation on $\mathcal{G}'$.

5 Experiments

Setup. We evaluate SHADOW-GNN on seven graphs. Six of them are for the node classification task: `Flickr` [53], `Reddit` [10], `Yelp` [53], `ogbn-arxiv`, `ogbn-products` and `ogbn-papers100M` [14]. The remaining is for the link prediction task: `ogbl-collab` [14]. The sizes of the seven graphs range from 9K nodes (`Flickr`) to 110M nodes (`ogbn-papers100M`). `Flickr`, `Reddit` and `Yelp` are under the inductive setting. `ogbn-arxiv`, `ogbn-products` and `ogbn-papers100M` are transductive. Consistent with the original papers, for the graphs on node classification, we measure "F1-micro" score for `Yelp` and "accuracy" for the remaining five graphs. For the link prediction task, we use "Hits@50" as the metric. See Appendix E.1 for details.

We construct SHADOW with six backbone models: GCN [21], GraphSAGE [10], GAT [45], JK-Net [50], GIN [49], SGC [47]. The first five are representatives of the state-of-the-art GNN architectures, jointly covering various message aggregation functions as well as the skip connection design. SGC simplifies normal GCN by moving all the neighbor propagation to the pre-processing step. Therefore, SGC is suitable for evaluating oversmoothing. The non-SHADOW models are trained with both full-batch and GraphSAINT-minibatch [53]. Due to the massive size of the full L-hop neighborhood, we need to perform sampling when training normal GNNs in order to make the computation time tractable. GraphSAINT is suitable for our purpose since 1. it is the state-of-the-art minibatch method which achieves high accuracy, and 2. it supports various GNN architectures and scales well to large graphs. On the other hand, for SHADOW-GNN, both training and inference are always executed in minibatches. One advantage of SHADOW-GNN is that the decoupling enables straightforward minibatch construction: each target just independently extracts the small subgraph on its own.

We implement two `EXTRACT` described in Section 3.4: 1. "PPR", where we set the node budget K as $\{200, 400\}$ for the largest `ogbn-papers100M` and $K \leq 200$ for all other graphs; 2. "L-hop", where we set the depth as $\{1, 2\}$. As for the model depth, most of our evaluation is on $L' \in \{3, 5\}$. Experiments on larger L' is deferred to Appendix F.6. All accuracy are measured by five runs without fixing random seeds. Hyperparameter tuning and architecture configuration are in Appendix E.4.

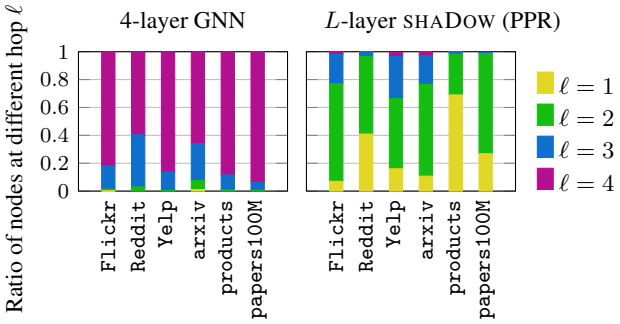


Figure 1: Neighbor composition of normal and SHADOW GNN

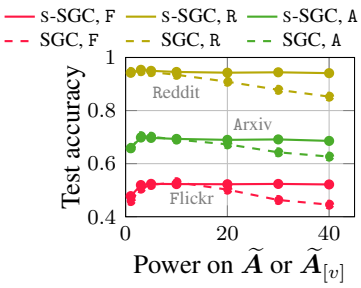


Figure 2: SGC oversmoothing

Table 1: Comparison on test accuracy / F1-micro score and inference cost (tuned with DropEdge)

Method	Layers	Flickr		Reddit		Yelp		ogbn-arxiv		ogbn-products	
		Accuracy	Cost	Accuracy	Cost	F1-micro	Cost	Accuracy	Cost	Accuracy	Cost
GCN	3	0.5159±0.0017	2E0	0.9532±0.0003	6E1	0.4028±0.0019	2E1	0.7170±0.0026	1E1	0.7567±0.0018	5E0
	5	0.5217±0.0016	2E2	0.9495±0.0012	1E3	OOM	1E3	0.7186±0.0017	1E3	OOM	9E2
GCN + GraphSAINT-RW	3	0.5155±0.0027	2E0	0.9523±0.0003	6E1	0.5110±0.0012	2E1	0.7093±0.0003	1E1	**0.8003**±0.0024	5E0
	5	0.5165±0.0026	2E2	0.9523±0.0012	1E3	0.5012±0.0021	1E3	0.7039±0.0020	1E3	0.7992±0.0021	9E2
SHADOW-GCN +PPR	3	0.5262±0.0018	**(1)**	0.9581±0.0004	**(1)**	0.5255±0.0012	**(1)**	0.7192±0.0025	**(1)**	0.7778±0.0030	**(1)**
	5	**0.5270**±0.0024	1E0	**0.9583**±0.0002	1E0	**0.5272**±0.0018	2E0	**0.7207**±0.0030	2E0	0.7844±0.0029	2E0
GraphSAGE	3	0.5140±0.0014	3E0	0.9653±0.0002	5E1	0.6178±0.0033	2E1	0.7192±0.0027	1E1	0.7858±0.0013	4E0
	5	0.5154±0.0052	2E2	0.9626±0.0004	1E3	OOM	2E3	0.7193±0.0037	1E3	OOM	1E3
GraphSAGE + GraphSAINT-RW	3	0.5176±0.0032	3E0	0.9671±0.0003	5E1	0.6453±0.0011	2E1	0.7107±0.0003	1E1	0.7923±0.0023	4E0
	5	0.5201±0.0032	2E2	0.9670±0.0010	1E3	0.6394±0.0003	2E3	0.7013±0.0021	1E3	0.7964±0.0022	1E3
SHADOW-SAGE + 2-hop	3	0.5288±0.0014	1E0	0.9660±0.0003	1E0	0.6493±0.0001	1E0	0.7163±0.0012	1E0	0.7993±0.0012	1E0
	5	0.5338±0.0038	2E0	0.9661±0.0002	2E0	0.6503±0.0001	2E0	0.7183±0.0012	2E0	0.8014±0.0020	2E0
SHADOW-SAGE + PPR	3	0.5344±0.0028	**(1)**	**0.9693**±0.0002	**(1)**	**0.6512**±0.0002	**(1)**	0.7234±0.0032	**(1)**	0.7945±0.0021	**(1)**
	5	**0.5424**±0.0025	2E0	0.9691±0.0003	2E0	0.6502±0.0001	2E0	**0.7255**±0.0013	2E0	**0.8043**±0.0026	2E0
GAT	3	0.5070±0.0032	2E1	OOM	3E2	OOM	2E2	0.7201±0.0011	1E2	OOM	3E1
	5	0.5164±0.0033	2E2	OOM	2E3	OOM	2E3	OOM	3E3	OOM	2E3
GAT + GraphSAINT-RW	3	0.5225±0.0053	2E1	0.9671±0.0003	3E2	0.6459±0.0002	2E2	0.6977±0.0003	1E2	0.8027±0.0028	3E1
	5	0.5153±0.0034	2E2	0.9651±0.0024	2E3	0.6478±0.0012	2E3	0.6954±0.0013	3E3	0.7990±0.0072	2E3
SHADOW-GAT + PPR	3	**0.5383**±0.0032	**(1)**	0.9703±0.0010	**(1)**	**0.6549**±0.0002	**(1)**	0.7243±0.0011	**(1)**	0.8014±0.0012	**(1)**
	5	0.5342±0.0023	2E0	**0.9710**±0.0008	2E0	0.6537±0.0004	2E0	**0.7283**±0.0027	2E0	**0.8094**±0.0034	2E0

SHADOW-GNN neighborhood. For both normal and SHADOW GNNs, Figure 1 shows on average how many neighbors are L hops away from the target. For a normal GNN, the size of the neighborhood grows rapidly with respect to L, and the nodes 4 hops away dominate the entire neighborhood. For SHADOW-GNN using the `EXTRACT` corresponding to Table 1, most neighbors concentrate within 2 hops. A small number of nodes are 3 hops away and almost no nodes are 4 or more hops away. Importantly, not only does the composition of the two kinds of neighborhood differ significantly, but also the size of SHADOW-GNN receptive field is much smaller (see also Table 1). Such small subgraphs are essential SHADOW-GNN's high computation efficiency. We can ignore the very few distant neighbors ($L \geq 4$), and regard the (effective) depth of SHADOW-GNN subgraph as $L = 2$ or 3. Such practical values of L guide us to set the model depth L'. When $L' = 3$, we already have the model deeper than the subgraph. When further increasing to $L' = 5$, any pair of subgraph nodes will be able to exchange messages to each other at least twice. Therefore, we set $L' = 3$ and 5 for Table 1. We show in Appendix F.6 that in some cases, we can further improve accuracy when $L' > 5$.

Comparison with baselines. Table 1 shows the performance comparison of SHADOW-GNN with the normal GNNs. All models on all datasets have uniform hidden dimension of 256. We define the metric "inference cost" as the average amount of computation to generate prediction for one test node. Inference cost is a measure of computation complexity (see Appendix B for the calculation) and is independent of hardware / implementation factors such as parallelization strategy, batch processing, *etc*. For SHADOW-GNN, we do not include the overhead on computing `EXTRACT`, since it is hard to calculate such cost analytically. Empirically, subgraph extraction is much cheaper than model computation (see Figure 6, 7 for time evaluation on CPU and GPU). During training, we apply DropEdge [38] to both the baseline and SHADOW models. DropEdge helps improve the baseline accuracy by alleviating oversmoothing, and benefits SHADOW-GNN due to its regularization effects. See Appendix F.2 for results on other architectures including GIN and JK-Net.

ACCURACY SHADOW-GNNs (with all neighborhood containing less than 200 nodes) achieve significantly higher accuracy across all graphs. Since we use the identical backbone architectures, the accuracy gains validate our "decoupling" principle. For 3 layers, SHADOW-GNNs are at least as good as, and sometimes better than the baseline. This indicates that a shallow neighborhood contains sufficient information, and customizing the receptive field benefits GNN accuracy (from Figure 1, a depth-3 $\mathcal{G}_{[v]}$ differs from the 3-hop neighborhood). Increasing to 5 layers, for normal GNNs, a deeper model sometimes results in accuracy degradation (even after DropEdge). For SHADOW-GNN, increasing the model depth is beneficial most of the time. Finally, the higher accuracy of the PPR `EXTRACT` compared with the 2-hop one demonstrates the importance of the extraction algorithm.

INFERENCE COST From Table 1, inference cost of SHADOW-GNN is orders of magnitude lower than the baselines (a 5-layer SHADOW-GNN is still much cheaper than a 3-layer normal GNN). The high cost of the baselines is due to the "neighborhood explosion" w.r.t. more layers. SHADOW-GNN is highly efficient and scalable as the cost only grows linearly with the model depth. Also, GraphSAINT only improves training efficiency as its inference operates on the full L-hop neighborhood.

LEADERBOARD COMPARISON For the two OGB graphs, `ogbn-arxiv` and `ogbn-products`, the highest ranking methods on leaderboard improves their accuracy by additional tricks like using node labels as the model input [15, 41] and data augmentation [24]. Our results in Table 1 achieve the same accuracy as the *rank-1* leaderboard models [28, 21, 33] which do not use those tricks.

Scaling to 100 million nodes. We further scale SHADOW-GNN to `ogbn-papers100M`, one of the largest public dataset. Even through the full graph size of `ogbn-papers100M` is two orders of magnitude larger than the graphs in Table 1, the local receptive field of SHADOW-GNN barely needs to increase. Since SHADOW-GNN performs minibatch computation, a low-end GPU with limited memory capacity can compute SHADOW-GNN on `ogbn-papers100M` efficiently. We show in Appendix that we can train and inference our model with as little as 2GB GPU memory consumption. This is clearly infeasible using normal GNNs. Table 2 summarizes our comparison with the top leaderboard methods. We only include those methods that do not use node labels as the model input (*i.e.*, the most standard setup). We achieve the *highest* test accuracy with at least 3 orders of magnitude reduction in neighborhood size. For SIGN-XL and SGC, their neighborhood is too large to count the exact size. Also, their preprocessing consumes $5\times$ more CPU memory than SHADOW-GNN (Appendix F.1).

Table 2: Leaderboard comparison on `papers100M`

Method	Test accuracy	Val accuracy	Neigh size
GraphSAGE+incep	0.6707±0.0017	0.7032±0.0011	4E5
SIGN-XL	0.6606±0.0019	0.6984±0.0006	> 4E5
SGC	0.6329±0.0019	0.6648±0.0020	> 4E5
SHADOW-GAT$_{200}$	0.6681±0.0016	0.7019±0.0011	2E2
SHADOW-GAT$_{400}$	**0.6710**±0.0015	**0.7067**±0.0012	3E2

Extending to link-level task. We further show that SHADOW-GNN is general and can be extended to the link prediction task. There are two settings of `ogbl-collab`. We follow the one where validation edges cannot be used in training updates. This is the setting which most leaderboard methods follow. Table 3 shows the comparison with the top GNN models under the same setting. SHADOW-SAGE outperforms the *rank-1* model with significant margin.

Table 3: Leaderboard comparison on `collab`

Method	Test Hits@50	Val Hits@50
SEAL	0.5371±0.0047	0.6389±0.0049
DeeperGCN	0.5273±0.0047	0.6187±0.0045
LRGA+GCN	0.5221±0.0072	0.6088±0.0059
SHADOW-SAGE	**0.5482**±0.0046	**0.6503**±0.0054

Oversmoothing. To validate Theorem 3.2, we pick SGC as the backbone architecture. SGC with power L is equivalent to L-layer GCN without activation. Performance comparison between SGC and SHADOW-SGC thus reveals the effect of oversmoothing without introducing other factors due to optimizing deep neural networks (e.g., vanishing gradients). In Figure 2, we vary the power of SGC and SHADOW-SGC from 1 to 40 (see Appendix E.5 for details). While SGC gradually collapses local information into global "white noise", accuracy of SHADOW-SGC does not degrade. This validates our theory that extracting local subgraphs prevents oversmoothing.

6 Conclusion

We have presented a design principle to decouple the depth and scope (*i.e.*, receptive field) of GNNs. Applying such principle on various GNN architectures can improve expressivity and computation efficiency of the corresponding model simultaneously. We have presented thorough theoretical analysis on expressivity from three different perspectives, and also rich design components (*e.g.*, subgraph extraction functions, architecture extensions) to implement such design principle.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. *arXiv preprint arXiv:1905.00067*, 2019.

[2] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

[3] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning*, pages 874–883. PMLR, 2020.

[4] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 2464–2473, New York, NY, USA, 2020. Association for Computing Machinery.

[5] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction, 2017.

[6] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

[7] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 257–266, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1393–1403, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. SIGN: Scalable inception graph neural networks, 2020.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[11] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[12] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian graph neural networks with adaptive connection sampling, 2020.

[13] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[14] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs, 2020.

[15] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks, 2020.

[16] Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general Graph Convolutional Networks, 2020.

[17] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Advances in neural information processing systems*, pages 4558–4567, 2018.

[18] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543, 2002.

[19] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[22] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019.

[23] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning, 2019.

[24] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks, 2020.

[25] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743. PMLR, 2019.

[26] David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

[27] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[28] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All you need to train deeper gcns, 2020.

[29] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into Graph Convolutional Networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*, 2018.

[30] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Aug 2020.

[31] Yu-Chen Lo, Stefano E Rensi, Wen Torng, and Russ B Altman. Machine learning in chemoinformatics and drug discovery. *Drug discovery today*, 23(8):1538–1546, 2018.

[32] Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. Break the ceiling: Stronger multi-scale deep graph convolutional networks. In *Advances in Neural Information Processing Systems 32*, pages 10945–10955. Curran Associates, Inc., 2019.

[33] Zhewei Wei Ming Chen, Bolin Ding Zengfeng Huang, and Yaliang Li. Simple and deep graph convolutional networks. 2020.

[34] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pages 3697–3707, 2017.

[35] Kenta Oono and Taiji Suzuki. Graph Neural Networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

[36] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. PinnerSage: Multi-modal user embedding framework for recommendations at Pinterest. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Aug 2020.

[37] Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 596–606, 2019.

[38] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards deep Graph Convolutional Networks on node classification. In *International Conference on Learning Representations*, 2020.

[39] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[40] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[41] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2020.

[42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[43] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackerman, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

[44] Chuxiong Sun and Guoshi Wu. Adaptive graph diffusion networks with hop-wise attention, 2020.

[45] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

[46] Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. AM-GCN: Adaptive multi-channel graph convolutional networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1243–1253, New York, NY, USA, 2020. Association for Computing Machinery.

[47] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

[48] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

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

[50] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks, 2018.

[51] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[52] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17009–17021. Curran Associates, Inc., 2020.

[53] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

[54] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

[55] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[56] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[57] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Efficient probabilistic logic reasoning with graph neural networks, 2020.

[58] Lingxiao Zhao and Leman Akoglu. PairNorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020.

[59] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the released code.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the released code. It contains all configurations.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See citation after each dataset.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]