SIGN: Scalable Inception Graph Neural Networks

Fabrizio Frasca[*]
Twitter / Imperial College London
United Kingdom

Emanuele Rossi[*]
Twitter / Imperial College London
United Kingdom

Davide Eynard
Twitter
United Kingdom

Ben Chamberlain
Twitter
United Kingdom

Michael M. Bronstein
Twitter / Imperial College London
United Kingdom

Federico Monti
Twitter
United Kingdom

Abstract

Graph representation learning has recently been applied to a broad spectrum of problems ranging from computer graphics and chemistry to high energy physics and social media. The popularity of graph neural networks has sparked interest, both in academia and in industry, in developing methods that scale to very large graphs such as Facebook or Twitter social networks. In most of these approaches, the computational cost is alleviated by a sampling strategy retaining a subset of node neighbors or subgraphs at training time. In this paper we propose a new, efficient and scalable graph deep learning architecture which sidesteps the need for graph sampling by using graph convolutional filters of different size that are amenable to efficient precomputation, allowing extremely fast training and inference. Our architecture allows using different local graph operators (e.g. motif-induced adjacency matrices or Personalized Page Rank diffusion matrix) to best suit the task at hand. We conduct extensive experimental evaluation on various open benchmarks and show that our approach is competitive with other state-of-the-art architectures, while requiring a fraction of the training and inference time. Moreover, we obtain state-of-the-art results on `ogbn-papers100M`, the largest public graph dataset, with over 110 million nodes and 1.5 billion edges.

1 Introduction

Deep learning on graphs, also known as *geometric deep learning* (GDL) [9] or *graph representation learning* (GRL) [25, 5, 66], has emerged in a matter of just a few years from a niche topic to one of the most prominent fields in machine learning. Graph deep learning models have recently scored successes in various applications relying on modeling relational data, see e.g. [65, 49, 42, 14, 17, 20, 47, 68, 58, 19, 51, 43]. Graph Neural Networks (GNNs) seek to generalize classical convolutional architectures (CNNs) to graph-structured data, with a wide variety of convolution-like operations available in the literature [52, 15, 3, 45, 54, 42, 34, 59, 57, 24].

Until recently, most of the research in the field has focused on small-scale datasets, and relatively little effort has been devoted to scaling these methods to web-scale graphs such as the Facebook or Twitter social networks. Scaling is a major challenge precluding the wide application of graph deep learning methods in industrial settings. Compared to Euclidean neural networks where the training loss can be decomposed into individual samples and computed independently, graph convolutional networks diffuse information between nodes along the edges of the graph, making the loss computation interdependent for different nodes. Furthermore, in typical graphs the number of nodes grows

[*]Equal contribution

Preprint. Under review.

exponentially with the increase of the filter receptive field, incurring significant computational and memory complexity. So far, various *graph sampling* approaches [24, 62, 11, 30, 12, 13, 64, 70] have been proposed as a way to alleviate the cost of training graph neural networks by selecting a small number of neighbors that reduce the computational and memory complexity.

In this paper, we take a different approach for scalable deep learning on graphs. We propose *SIGN*, a simple scalable Graph Neural Network architecture inspired by the inception module [56, 32]. *SIGN* combines graph convolutional filters of different types and sizes that are amenable to efficient precomputation, allowing extremely fast training and inference with complexity independent of the graph structure. Our architecture is able to scale to web-scale graphs without resorting to any sample technique and retaining sufficient expressiveness for effective learning: while being faster in training and, especially, inference (even one order of magnitude speedup), by employing *SIGN* with only one graph convolutional layer we are able to achieve results on par with state-of-the-art on several large-scale graph learning datasets. In particular, *SIGN* obtains state-of-the-art results on `ogbn-papers100M`, the largest public graph learning benchmark, with over 110 million nodes and 1.5 billion edges.

These results raise the important question on when deep graph neural network architectures are useful, especially when scalability is an important requirement, as in large-scale industrial systems. Significant effort has recently been devoted to methods allowing to design deep Graph Neural Networks with many graph convolutional layers [61, 21, 40, 67, 50], which otherwise appear difficult to train [41, 35, 60]. While there is strong evidence in favor of depth on geometric graphs [28, 40, 21], there has been almost no gain from depth on general irregular graphs like 'small-world' networks [53, 50]. Given the abundance of such graphs e.g. in social network applications, it is important to take a step back and deliberate if deep architectures are the right approach. We conjecture that deep graph learning architectures are not useful for general irregular graphs and argue that future research in the field should focus on designing local more expressive operators [4, 44, 18] rather than going deeper.

2 Background and Related Work

2.1 Deep learning on graphs

The goal of graph representation learning is to construct an embedding representing the structure of the graph and the data thereon. In node-wise prediction problems, we distinguish between **Transductive** setting, which assumes that the entire graph is known, and thus the same graph is used during training and testing (albeit different nodes are used for training and testing), and **Inductive** setting, in which training and testing are performed on different graphs. A typical graph neural network architecture consists of graph **Convolution-like operators** (discussed in Section 2.3) performing local aggregation of features by means of message passing with the neighbor nodes, and possibly **Pooling** amounting to fixed [16] or learnable [63, 8] graph coarsening. Additionally, graph **Sampling** schemes (detailed in Section 2.4) can be employed on large-scale graphs to reduce the computational complexity.

2.2 Basic notions

Let $\mathcal{G} = (\mathcal{V} = \{1, \ldots, n\}, \mathcal{E}, \mathbf{W})$ be an undirected weighted graph, represented by the symmetric $n \times n$ *adjacency matrix* $\mathbf{W}$, where $w_{ij} > 0$ if $(i, j) \in \mathcal{E}$ and zero otherwise. The diagonal *degree matrix* $\mathbf{D} = \mathrm{diag}(\sum_{j=1}^{n} w_{1j}, \ldots, \sum_{j=1}^{n} w_{nj})$ represents the number of neighbors of each node. We further assume that each node is endowed with a d-dimensional feature vector and arrange all the node features as rows of the $n \times d$-dimensional matrix $\mathbf{X}$. We denote by $\mathbf{A} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ the normalized adjacency matrix. The normalized *graph Laplacian* is an $n \times n$ positive semi-definite matrix $\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$.

2.3 Convolution-like operators on graphs

Spectral methods Bruna et al. [10] used the analogy between the eigenvectors of the graph Laplacian and the Fourier transform to generalize convolutional neural networks (CNN) [37] to graphs. Among the key drawbacks of this approach is a high (at least $\mathcal{O}(n^2)$ complexity), large

($\mathcal{O}(n)$) number of filter parameters, no spatial localization, and no generalization of filters across graphs. Furthermore, the method explicitly assumes the underlying graph to be undirected, in order for the Laplacian to be a symmetric matrix with orthogonal eigenvectors.

ChebNet A way to address the shortcomings of [10] is to model the filter as a transfer function $\hat{g}(\lambda)$, applied to the Laplacian as $\hat{g}(\boldsymbol{\Delta}) = \boldsymbol{\Phi}\hat{g}(\boldsymbol{\Lambda})\boldsymbol{\Phi}^\top$. Filters computed in this manner are stable under graph perturbations [38]. In the case when $\hat{g}$ is expressed as simple matrix-vector operations (e.g. a polynomial [15] or rational function [39]), the eigendecomposition of the Laplacian can be avoided altogether. A particularly simple choice is a polynomial spectral filter $\hat{g}(\lambda) = \sum_{k=0}^{r} \theta_k \lambda^k$ of degree r, allowing the convolution to be computed entirely in the spatial domain as

$$\mathbf{Y} = \hat{g}(\boldsymbol{\Delta})\mathbf{X} = \sum_{k=0}^{r} \theta_k \boldsymbol{\Delta}^k \mathbf{X}. \tag{1}$$

with $\mathcal{O}(r)$ parameters $\theta_0, \dots, \theta_r$, does not require explicit multiplication by $\boldsymbol{\Phi}$, and has a compact support of r hops in the node domain. Though originating from a spectral construction, the resulting filter is an operation in the node domain amounting to a successive aggregation of features in the neighbor nodes, which can be performed with complexity $\mathcal{O}(|\mathcal{E}|r) \approx \mathcal{O}(nr)$. The polynomial filters can be combined with non-linearities, concatenated in multiple layers, and interleaved with pooling layers based on graph coarsening [15]. The Laplacian in (1) can be replaced with other operators that diffuse information across neighbor nodes, e.g. the simple or normalized adjacency matrix, without affecting performance.

GCN In the case $r = 1$, equation (1) reduces to computing $(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2})\mathbf{X}$, which can be interpreted as a combination of the node features and the neighbors filtered features. Kipf and Welling [34] proposed a model of graph convolutional networks (*GCN*) combining node-wise and graph diffusion operations:

$$\mathbf{Y} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{W}}\tilde{\mathbf{D}}^{-1/2}\mathbf{X}\boldsymbol{\Theta} = \tilde{\mathbf{A}}\mathbf{X}\boldsymbol{\Theta}. \tag{2}$$

Here $\tilde{\mathbf{W}} = \mathbf{I} + \mathbf{W}$ is the adjacency matrix with self-loops, $\tilde{\mathbf{D}} = \mathrm{diag}(\sum_{j=1}^{n} \tilde{w}_{1j}, \dots, \sum_{j=1}^{n} \tilde{w}_{nj})$ is the respective degree matrix, and $\boldsymbol{\Theta}$ is a matrix of learnable parameters.

S-GCN Stacking L *GCN* layers with element-wise non-linearity σ and a final layer for node classification with activation ξ (e.g. softmax or sigmoid), it is possible to obtain filters with larger receptive fields on the graph nodes,

$$\mathbf{Y} = \xi(\tilde{\mathbf{A}} \cdots \sigma(\tilde{\mathbf{A}}\mathbf{X}\boldsymbol{\Theta}^{(1)}) \cdots \boldsymbol{\Theta}^{(L)}).$$

Wu et al. [59] argued that graph convolutions with large filters is practically equivalent to multiple convolutional layers with small filters and showed that all but the last non-linearities can be removed without practically harming the performance, resulting in the *simplified GCN* (*S-GCN*) model,

$$\mathbf{Y} = \xi(\tilde{\mathbf{A}}^L\mathbf{X}\boldsymbol{\Theta}^{(1)} \cdots \boldsymbol{\Theta}^{(L)}) = \xi(\tilde{\mathbf{A}}^L\mathbf{X}\boldsymbol{\Theta}). \tag{3}$$

MotifNet Monti et al. [44] used adjacency matrices with weights proportional to the count of simple subgraphs (motifs) on edges in order to account for higher order structures. Related ideas have been explored using higher-order Laplacians on simplicial complexes [4].

2.4 Graph sampling

For Web-scale graphs such as Facebook or Twitter that typically have $n = 10^8 \sim 10^9$ nodes and $|\mathcal{E}| = 10^{10} \sim 10^{11}$ edges, the diffusion matrix cannot be stored in memory for training, making the straightforward application of graph neural networks impossible. Graph sampling has been shown to be a successful technique to scale GNNs to large graphs by approximating local connectivity with subsampled versions which are amenable for computation.

Node-wise sampling These strategies perform graph convolutions on *partial* node neighborhoods to reduce computational and memory complexity, and are coupled with minibatch training, where each training step is performed only on a batch of nodes rather than on the whole graph. A training batch

Table 1: Theoretical time complexity where $L_{\rm c}, L_{\rm ff}$ is the number of graph convolution and MLP layers, r is the filter size, N the number of nodes (in training or inference, respectively), $|\mathcal{E}|$ the number of edges, and d the feature dimensionality (assumed fixed for all layers). For *GraphSAGE*, k is the number of sampled neighbors per node. Forward pass complexity corresponds to an entire epoch where all nodes are seen.

	Preproc.	*Forward Pass*
GraphSAGE	$\mathcal{O}(k^{L_{\rm c}}N)$	$\mathcal{O}(k^{L_{\rm c}}Nd^2)$
ClusterGCN	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	$\mathcal{O}(L_{\rm c}\lvert\mathcal{E}\rvert d + L_{\rm ff}Nd^2)$
GraphSAINT	$\mathcal{O}(kN)$	$\mathcal{O}(L_{\rm c}\lvert\mathcal{E}\rvert d + L_{\rm ff}Nd^2)$
SIGN-r	$\mathcal{O}(r\lvert\mathcal{E}\rvert d)$	$\mathcal{O}(rL_{\rm ff}Nd^2)$

is assembled by first choosing b 'optimization' nodes and partially expanding their corresponding neighborhoods. In a single training step, the loss is computed and optimized only for optimization nodes. Node-wise sampling coupled with minibatch training was first introduced in *GraphSAGE* [24] to address the challenges of scaling GNNs. PinSAGE [62] extended *GraphSAGE* by exploiting a neighbor selection method using scores from approximations of Personalized PageRank [26] via random walks. *VR-GCN* [12] uses control variates to reduce the variance of stochastic training and increase the speed of convergence with a small number of neighbors.

Layer-wise sampling A characteristic of many graphs, in particular 'small-world' social networks, is the exponential growth of the neighborhood size with number of hops L. [11, 30] avoid over-expansion of neighborhoods to overcome the redundancy of node-wise sampling. Nodes in each layer only have directed edges towards nodes of the next layer, thus bounding the maximum amount of computation to $\mathcal{O}(b^2)$ per layer. Moreover, sharing common neighbors prevents feature replication across the batch, drastically reducing the memory complexity during training.

Graph-wise sampling In [13, 64], feature sharing is further advanced: each batch consists of a connected subgraph and at each training iteration the GNN model is optimized over all nodes in the subgraph. In *ClusterGCN* [13], non-overlapping clusters are computed as a pre-processing step and then sampled during training as input minibatches. *GraphSAINT* [64] adopts a similar approach, while also correcting for the bias and variance of the minibatch estimators when sampling subgraphs for training. It also explores different schemes to sample the subgraphs such as a random walk-based sampler, which is able to co-sample nodes having high influence on each other and guarantees each edge has a non-negligible probability of being sampled.

3 Scalable Inception Graph Neural Networks

In this work we propose *SIGN*, an alternative method to scale Graph Neural Networks to very large graphs. The key building block of our architecture is a set of linear diffusion operators represented as $n \times n$ matrices $\mathbf{A}_1, \ldots, \mathbf{A}_r$, whose application to the node-wise features can be pre-computed. For node-wise classification tasks, our architecture has the form (Figure 1):

$$\begin{aligned} \mathbf{Z} &= \sigma\left([\mathbf{X}\mathbf{\Theta}_0, \mathbf{A}_1\mathbf{X}\mathbf{\Theta}_1, \ldots, \mathbf{A}_r\mathbf{X}\mathbf{\Theta}_r]\right) \\ \mathbf{Y} &= \xi\left(\mathbf{Z}\mathbf{\Omega}\right), \end{aligned} \tag{4}$$

where $\mathbf{\Theta}_0, \ldots, \mathbf{\Theta}_r$ and $\mathbf{\Omega}$ are learnable matrices respectively of dimensions $d \times d'$ and $d'(r+1) \times c$ for c classes, and σ, ξ are non-linearities, the second one computing class probabilities, e.g. via softmax or sigmoid function, depending on the task at hand. We denote a model with r operators by *SIGN-r*.

A key observation is that matrix products $\mathbf{A}_1\mathbf{X}, \ldots, \mathbf{A}_r\mathbf{X}$, in equation (4) *do not depend* on the learnable model parameters and can be easily precomputed. For large graphs, distributed computing infrastructures such as Apache Spark can speed up computation. This effectively reduces the computational complexity of the overall model to that of a Multi-Layer Perceptron (MLP), i.e. $\mathcal{O}(rL_{\rm ff}Nd^2)$, where d is the number of features, N the number of nodes in the training/testing graph and $L_{\rm ff}$ is the overall number of feed-forward layers in the model.

Table 1 compares the complexity of our *SIGN* model to the other scalable architectures *GraphSAGE*, *ClusterGCN*, and *GraphSAINT*. While all models scale linearly w.r.t. the number of nodes N, we

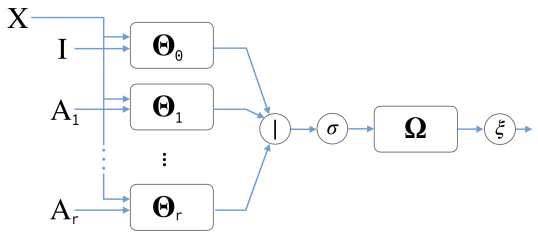

Figure 1: The *SIGN* architecture for r generic graph filtering operators. $\mathbf{\Theta}_k$ represents the k-th dense layer transforming node-wise features downstream the application of operator k, $|$ is the concatenation operation and $\mathbf{\Omega}$ refers to the dense layer used to compute final predictions.

Table 2: By appropriate configuration, *SIGN* inception layer is able to replicate some popular graph convolutional layers. α represents the learnable parameter of a PReLU activation.

	$\mathbf{B}_1, \dots, \mathbf{B}_r$	α	$\mathbf{\Theta}_0, \dots, \mathbf{\Theta}_r$	$\mathbf{\Omega}$
ChebNet [15]	$\mathbf{\Delta}, \dots, \mathbf{\Delta}^r$	1	$\mathbf{\Theta}_0, \dots, \mathbf{\Theta}_r$	$[\mathbf{I}, \dots, \mathbf{I}]^\top$
GCN [34]	$r = 1,\ \tilde{\mathbf{A}}$	1	$\mathbf{0}, \mathbf{\Theta}$	$[\mathbf{0}, \mathbf{I}]^\top$
S-GCN [59]	$r = 1,\ \tilde{\mathbf{A}}^L$	1	$\mathbf{0}, \mathbf{\Theta}$	$[\mathbf{0}, \mathbf{I}]^\top$

show experimentally that our model is significantly faster than all others due to the fact the forward and backward pass complexity of our model does not depend on the graph structure. Unlike the aforementioned scalable methods, *SIGN* is not based on sampling nodes or subgraphs, operations potentially introducing bias into the optimization procedure.

Inception-like module Within the *SIGN* framework, it is possible to choose one specific operator $\mathbf{B}$ and to define $\mathbf{A}_k = \mathbf{B}^k$ for $k = 1, \dots, r$. In this setting the proposed model is analogous to the popular *Inception module* [56] for classic CNN architectures: it consists of convolutional filters of different sizes determined by the parameter r, where $r = 0$ corresponds to 1×1 convolutions in the inception module (amounting to linear transformations of the features in each node without diffusion across nodes). Owing to this analogy, we refer to our model as the *Scalable Inception Graph Network* (*SIGN*). It is also easy to observe that various graph convolutional layers can be obtained as particular settings of (4). In particular, by setting the σ non-linearity to PReLU [27], ChebNet, GCN, and *S-GCN* can be automatically learnt if suitable diffusion operator $\mathbf{B}$ and activation ξ are used (see Table 2).

Choice of the operators Generally speaking, the choice of the diffusion operators jointly depends on the task, graph structure, and the features. In complex networks such as social graphs, operators induced by triangles or cliques might help distinguishing edges representing weak or strong ties [22]. In graphs with noisy connectivity, it was shown that diffusion operators based on Personalized PageRank (PPR) or Heat Kernel can boost performance [36]. In our experiments, we choose three specific types of operators: simple (normalized) adjacency, Personalized PageRank-based adjacency, and triangle-based adjacency matrices, as well as their powers. We denote by *SIGN(p,s,t)* with $r = p + s + t$ the configuration using up to the p-th, s-th, and t-th power of simple GCN-normalized, PPR-based, and triangle-based adjacency matrices, respectively. Lastly, when working on directed graphs, *SIGN* can be equipped with powers of the (properly normalized) directed adjacency matrix $\mathbf{W}_d$ and its transpose $\mathbf{W}_d^\top$, in addition to the standard operators built on top of its undirected counterpart $\mathbf{W} = \frac{1}{2}(\mathbf{W}_d + \mathbf{W}_d^\top)$.

SIGN and S-GCN *SIGN* and *S-GCN* are the only graph neural models which are inherently 'shallow': contrary to standard GNN architectures, graph convolutional layers are not sequentially stacked, but either collapsed into a single linear filtering operation (S-GCN) or applied in parallel to obtain multi-scale node representations capturing diverse connectivity patterns depending on the chosen operators (*SIGN*). This is the crucial feature that allows these models to naturally scale their training and inference to graphs of any size and family given that all graph operations can be conveniently pre-computed. While we notice that *S-GCN* can be considered as a specific configuration of *SIGN*-1 (it is sufficient to choose $\mathbf{A}_1 = \tilde{\mathbf{A}}^L$ and to constrain $\mathbf{\Theta}_0$ to $\mathbf{0}$, see Table 2), we remark

the fact that the more general *SIGN* architecture easily allows to incorporate more expressivity via parallel application of several, possibly, domain-specific, operators. We experimentally show this in the following section where we demonstrate that the *S-GCN* paradigm is indeed too limiting and that a more expressive *SIGN* model is not only able to perform on par with 'deeper' sampling-based models, but also to achieve state-of-the-art results on the largest publicly available graph learning benchmark.

4 Experiments

Datasets We evaluated the proposed method on node-wise classification tasks, both in transductive and inductive settings. Inductive experiments are performed using four datasets: `Reddit` [24], `Flickr`, `Yelp` [64], and `PPI` [69]. To date, these are the largest graph learning inductive node classification benchmarks available in the public domain. Related tasks are multiclass node-wise classification for `Reddit` and `Flickr` and multilabel classification for `Yelp` and `PPI`. Transductive experiments were performed on the new `ogbn-products` and `ogbn-papers100M` datasets [29]. The former represents an Amazon product co-purchasing network [7] where the task is to predict the category of a product in a multi-class classification setup. The latter represents a directed citation network of ~ 111 million academic papers, where the task is to leverage information from the entire citation network to infer the labels (subject areas) of a smaller subset of ArXiv papers. Overall, this dataset is orders-of-magnitude larger than any existing node classification dataset and is therefore the most important testbed for the scalability of *SIGN* and related methods.

Furthermore, we also test the scalability of our method on `Wikipedia` links [1], a large-scale network of links between articles in the English version of Wikipedia.

Statistics for all the datasets are reported in Table 3.

Setup We tested several *SIGN*(p, s, t) configurations, with p the maximum power of the GCN-normalized adjacency matrix, s that of a random-walk normalized PPR diffusion operator [36], and t that of a row-normalized triangle-induced adjacency matrix [44], with weights proportional to edge occurrences in closed triads. PPR-based operators are computed from a symmetrically normalized adjacency transition matrix in an approximated form, with a restart probability of $\alpha = 0.01$ for inductive datasets and $\alpha = 0.05$ in the transductive case. To allow for larger model capacity in the inception modules and in computing final model predictions, we replace the single-layer projections performed by $\mathbf{\Theta}_i$ and $\mathbf{\Omega}$ modules with multiple feedforward layers. Model parameters are found by minimizing the cross-entropy loss via minibatch gradient descent with the Adam optimizer [33]. Early stopping is applied with a patience of 15. In order to limit overfitting, we apply the standard regularization techniques of weight decay and dropout [55]. Additionally, batch-normalization [31] was used in every layer to stabilize training and increase convergence speed. Architectural and optimization hyperparameters were estimated using Bayesian optimization with a tree Parzen estimator surrogate function [6] over all inductive datasets. As for the the transductive setting, we employ standard exhaustive search on a predefined hyperparameter grid on `ogbn-products`, while on `ogbn-papers100M` we only test a basic configuration that can be found in Supplemetary Materials, along with further details on the hyperparameter search spaces. Given that this dataset represents a directed network, we experimented with operators built via asymmetric normalization of the original directed adjacency matrix and its transpose, as well as their powers. The `Wikipedia` dataset, due to the lack of node attributes and labels, is only used to assess scalability: to this end, we randomly generate 100-dimensional node feature vectors and scalar targets and consider the whole network for both training and inference. No hyperparameter tuning is required in this case.

Baselines On the inductive datasets, we compare our method with *GCN* [34], *FastGCN* [11], *Stochastic-GCN* [12], *AS-GCN* [30], *GraphSAGE* [24], *ClusterGCN* [13], and *GraphSAINT* [64], which constitute the current state-of-the-art. On `ogbn-products` we compare against scalable sampling-free baselines, i.e. a feed-forward network trained over node features only (*MLP*) and on their concatenation with structural *Node2Vec* embeddings [23], and the sampling-based approaches *ClusterGCN* [13] and *GraphSAINT* [64]. As for `ogbn-papers100M`, *SIGN* is compared with sampling-free baselines: an *MLP* trained on node features and an *S-GCN* model. Sampling-based methods have not been scaled yet to this benchmark. All results for OGB datasets are directly taken from the latest version of the arXiv paper [29] (*v4*, at the time of writing). Lastly, being *S-GCN* an

Table 3: Summary of (s)ingle and (m)ulti-label dataset statistics. `Wikipedia` is used, with random features, for timing purposes only.

	n	$\lvert\mathcal{E}\rvert$	*Avg. Deg.*	d	Classes	Train / Val / Test
`Wikipedia`	12,150,976	378,142,420	62	100	2(s)	100% / — / 100%
`ogbn-papers100M`	111,059,956	1,615,685,872	30	128	172(s)	78% / 8% / 14%
`ogbn-products`	2,449,029	61,859,140	51	100	47(s)	10% / 2% / 88%
`Reddit`	232,965	11,606,919	50	602	41(s)	66% / 10% / 24%
`Yelp`	716,847	6,977,410	10	300	100(m)	75% / 10% / 15%
`Flickr`	89,250	899,756	10	500	7(s)	50% / 25% / 25%
`PPI`	14,755	225,270	15	50	121(m)	66% / 12% / 22%

Table 4: Mean and standard deviation of preprocessing, training (one epoch) and inference times, in seconds, on `ogbn-products` and `Wikipedia` datasets, computed over 10 runs. *SIGN-*r denotes architecture with r precomputed operators. Preprocessing and training times for *ClusterGCN* on `Wikipedia` are not reported due to the clustering algorithm failing to complete.

	`ogbn-products`			`Wikipedia`		
	Preprocessing	*Training*	*Inference*	*Preprocessing*	*Training*	*Inference*
ClusterGCN	36.93 ± 0.52	13.34 ± 0.16	93.00 ± 0.68	—	—	183.76 ± 3.01
GraphSAINT	52.06 ± 0.54	2.89 ± 0.05	94.76 ± 0.81	123.60 ± 1.60	135.73 ± 0.06	209.86 ± 4.73
SIGN-2	88.21 ± 1.33	1.04 ± 0.10	2.86 ± 0.10	192.88 ± 0.12	62.37 ± 0.17	13.40 ± 0.15
SIGN-4	160.16 ± 1.20	1.54 ± 0.04	3.79 ± 0.08	326.21 ± 1.14	93.84 ± 0.08	18.15 ± 0.05
SIGN-6	226.48 ± 1.43	2.05 ± 0.00	4.84 ± 0.08	459.24 ± 0.14	125.24 ± 0.03	22.94 ± 0.02
SIGN-8	297.92 ± 2.92	2.53 ± 0.04	5.88 ± 0.09	598.67 ± 0.82	154.73 ± 0.12	27.69 ± 0.11

important baseline for our model, we additionally report its performance on all the other datasets as well. In this case we choose power L of its (only) operator $\mathbf{A}^L$ as the value p of the best corresponding *SIGN(p,s,t)* configuration and we tune its hyperparameters in the same space searched for *SIGN*.

Implementation *SIGN* is implemented using Pytorch [48]. All experiments, including timings, were run on an AWS p2.8xlarge instance, with 8 NVIDIA K80 GPUs, 32 vCPUs, a processor Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz and 488GiB of RAM.

4.1 Results

Inductive Table 5 presents the results on the inductive dataset. In line with [64], we report the micro-averaged F1 score means and standard deviations computed over 10 runs. For each dataset we report the best performing *SIGN* configuration, specifying the maximum power for each of the three employed operators. *SIGN* outperforms other methods on `Reddit` and `Flickr`, and performs competitively to state-of-the-art on `PPI`. Our performance on `Yelp` is worse than in the other datasets; we hypothesize that a more tailored choice of operators is required to better suit the characteristics of this dataset. Interestingly, *SIGN* significantly outperforms *S-GCN* in all datasets, suggesting that the additional expressivity introduced by the different operators in our model is required for effective learning.

Transductive *SIGN* obtains state-of-the-art results on the `ogbn-papers100M` dataset (Table 8), outperforming other sampling-free methods by at least 1.8%. This shows that *SIGN* can scale to massive graphs while retaining ample expressivity. Sampling based methods have not been scaled yet to this benchmark. In `ogbn-papers100M` only $\sim 1.35\%$ of nodes are labeled; at *each* training and inference iteration these methods still need to perform computation on subgraphs where the majority of nodes are unlabeled and thus do not contribute to the computation of loss and evaluation metrics. On the contrary, at training and inference *SIGN* only processes the required labeled nodes given that the graph has already been employed during the one-time pre-computation phase, thus avoiding this redundant computation and memory usage.

`ogbn-products` results are reported in Table 6. *SIGN* outperforms all other sampling-free methods by at least 2.7%. However, contrary to the inductive benchmarks, sampling methods outperform *SIGN* and appear to generally be more suitable to this dataset. We hypothesise that, on this particular task, sampling may implicitly act as a regularizer, making these methods generalize better to the held-out test set, which in this dataset is sampled from a different distribution w.r.t. training and

Table 5: Micro-averaged F1 scores. For *SIGN*, we show the best performing configurations. The top three performance scores are highlighted as: **First**, **Second**, **Third**.

	Reddit	Flickr	PPI	Yelp
GCN [34]	0.933±0.000	0.492±0.003	0.515±0.006	0.378±0.001
FastGCN [11]	0.924±0.001	**0.504±0.001**	0.513±0.032	0.265±0.053
Stochastic-GCN [12]	**0.964±0.001**	0.482±0.003	**0.963±0.010**	**0.640±0.002**
AS-GCN [30]	0.958±0.001	**0.504±0.002**	0.687±0.012	—
GraphSAGE [24]	0.953±0.001	0.501±0.013	0.637±0.006	**0.634±0.006**
ClusterGCN [13]	0.954±0.001	0.481±0.005	0.875±0.004	0.609±0.005
GraphSAINT [64]	**0.966±0.001**	**0.511±0.001**	**0.981±0.004**	**0.653±0.003**
S-GCN [59]	0.949±0.000	0.502±0.001	0.892±0.015	0.358±0.006
SIGN	**0.968±0.000**	**0.514±0.001**	**0.970±0.003**	0.631±0.003
(p, s, t)	$(4, 2, 0)$	$(4, 0, 1)$	$(2, 0, 1)$	$(2, 0, 1)$

validation nodes [29]. This phenomenon, as well as its connection to the DropEdge method [50] and the bottleneck problem [2], would be object of further investigation.

Runtime While performing on par or better than state-of-the-art methods on most benchmarks in terms of accuracy, our method has the advantage of being significantly faster than other methods for large graphs. We perform comprehensive timing comparisons on `ogbn-products` and `Wikipedia` datasets and report average training, inference, and preprocessing times in Table 4. For these experiments, we run the implementations of *ClusterGCN* and *GraphSAINT* provided in the OGB code repository[2].

We use these datasets rather than `ogbn-papers100M` so we can compare to *ClusterGCN* and *GraphSAINT*. For completeness we report, however, that on `ogbn-papers100M` the best performing *SIGN(3,3,3)* model completes one evaluation pass on the validation set in 1.99 ± 0.05 seconds and on the test set in 3.34 ± 0.04 seconds (statistics are estimated over 10 runs and include the time required by device data transfers and by the computation of evalution metric).

Our model is faster than *ClusterGCN* and of comparable speed w.r.t. *GraphSAINT* in training[3], while being by far the fastest approach in inference: all *SIGN* architectures are always at least one order of magnitude faster than other methods, with the largest one (8 operators) requiring no more than 30 seconds to perform inference on over 12M nodes. *SIGN*'s preprocessing is slightly longer than other methods, but we notice that most of the calculations can be cast as sparse matrix multiplications and easily parallelized with frameworks for distributed computing. We envision to engineer faster and even more scalable *SIGN* preprocessing implementations in future developments of this work. Finally, in order to also study the convergence behavior of our proposed model, in Figure 2 we plot the validation performance on `ogbn-products` from the start of the training as a function of run time for *ClusterGCN*, *GraphSAINT* and several *SIGN* configurations. We observe that *SIGN* does not only converge to a better validation accuracy than other methods, but also exhibits much faster convergence than *ClusterGCN* and comparable speed than to *GraphSAINT*.

Ablation study How do different operator combinations affect *SIGN* performance? Results obtained with different choices of operators and their powers are reported in Tables 6, 8 and 7 for, respectively, the the transductive `ogbn-products` and `ogbn-papers100M` and inductive datasets. We notice that best performance is obtained on each benchmark by a specific combination of operators, remarking the fact that each dataset features particular topological and content characteristics requiring suitable filters. Interestingly, we also observe that while PPR operators do not bring significant improvements in the inductive setting (being even harmful in certain cases), they are beneficial on the transductive `ogbn-products`. This finding is in accordance with [36], where the effectiveness of PPR diffusion operators in transductive settings has been extensively studied. Finally, we notice promising results attained in `Flickr` and `PPI` inductive settings by pairing standard adjacency matrices with a triangle-induced one. Studying the effect of operators induced by more complex network motifs is left for future research.

[2] https://github.com/snap-stanford/ogb/tree/master/examples/nodeproppred/products

[3] Traning time is measured as forward-backward time to complete one epoch.

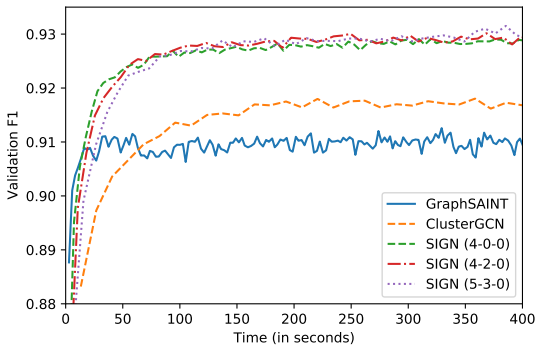

Figure 2: Convergence of different methods on `ogbn-products`.

Table 6: Performance on `ogbn-products`. *SIGN(p,s,t)* refers to a configuration using p, s, and t powers of simple, PPR-based, and triangle-based adjacency matrices. The top three performance scores are highlighted as: **First**, **Second**, **Third**.

	Training	*Validation*	*Test*
MLP	84.03±0.93	75.54±0.14	61.06±0.08
Node2Vec	93.39±0.10	90.32±0.06	72.49±0.10
S-GCN(L=5)	92.54±0.09	91.38±0.07	74.87±0.25
ClusterGCN	93.75±0.13	92.12±0.09	**78.97±0.33**
GraphSAINT	92.71±0.14	91.62±0.08	**79.08±0.24**
SIGN(3,0,0)	96.21±0.31	**92.99±0.05**	76.52±0.14
SIGN(3,0,1)	**96.46±0.29**	92.93±0.04	75.73±0.20
SIGN(3,3,0)	**96.87±0.23**	**93.02±0.04**	77.13±0.10
SIGN(5,0,0)	95.99±0.69	92.98±0.18	76.83±0.39
SIGN(5,3,0)	**96.92±0.46**	**93.10±0.08**	**77.60±0.13**

5 Conclusion and Future Work

In this paper we presented *SIGN*, a sampling-free Graph Neural Network model that is able to easily scale to gigantic graphs while retaining enough expressive power to attain competitive results in all large-scale graph learning benchmarks. *SIGN* attains state-of-the-art results on many of them, including the massive `ogbn-papers100M`, currently the largest publicly available node-classification benchmark with $\sim$ 111M nodes and $\sim$ 1.6B edges. Our experiments have further demonstrated that the ability of our model to flexibly incorporate diverse, possibly domain-specific, operators is crucial to overcome the expressivity limitations of other sampling-free scalable models such as *S-GCN*, which *SIGN* has constantly outperformed over all datasets. Overall, our architecture achieves an optimal trade-off between simplicity and expressiveness; as it has shown to attain competitive results with fast training and inference, it represents the most suitable architecture for scalable applications to web-scale graphs.

Depth vs. width for Graph Neural Networks Our results have shown that it is possible to obtain competitive – and often state-of-the-art – results with one single graph convolutional layer and hence a shallow architecture. An important question is, therefore, when one should apply deep architectures to graphs, where by 'depth' we refer to the number of stacked graph convolutional layers. Deep Graph Neural Networks are notoriously hard to train due to vanishing gradients and feature smoothing [41, 35, 60], and, although recent works have shown that these issues can be addressed to some extent [61, 21, 40, 67, 50], yet extensive experiments conducted in [50] showed that depth often does not bring any significant gain in performance w.r.t. to shallow baselines. A promising direction for future investigation is, rather than 'going deep', to 'go wide', in the sense of exploring more expressive local operators. We believe this to be especially crucial in all those settings where scalability is a concern of paramount importance, such as in industrial large-scale systems.

Extensions In our experiments, triangle-based operators showed promising results. Possible extensions can employ operators that account for higher-order structures such as simplicial complexes

Table 7: Impact of various operator combinations on inductive datasets. Best results are in bold.

	Reddit	Flickr	PPI	Yelp
SIGN(2,0,0)	0.966±0.003	0.503±0.003	0.965±0.002	0.623±0.005
SIGN(2,0,1)	0.966±0.000	0.510±0.001	**0.970±0.003**	**0.631±0.003**
SIGN(2,2,0)	0.967±0.000	0.495±0.002	0.964±0.003	0.617±0.005
SIGN(4,0,0)	0.967±0.000	0.508±0.001	0.959±0.002	0.623±0.004
SIGN(4,0,1)	0.967±0.000	**0.514±0.001**	0.965±0.003	0.623±0.004
SIGN(4,2,0)	**0.968±0.000**	0.500±0.001	0.930±0.010	0.618±0.004
SIGN(4,2,1)	0.967±0.000	0.508±0.002	0.969±0.001	0.620±0.004

Table 8: Results on `ogbn-papers100M`, the largest public graph dataset with over 110 million nodes. *SIGN(p,d,f)* refers to a configuration using p, d, and f powers of simple undirected, directed and directed-transposed adjacency matrices. The top three performance scores are highlighted as: **First**, **Second**, **Third**.

	Training	*Validation*	*Test*
MLP	54.84±0.43	49.60±0.29	47.24±0.31
Node2Vec	—	55.60±0.23	58.07±0.28
S-GCN(L=3)	67.54±0.43	66.48±0.20	63.29±0.19
SIGN(3,0,0)	**70.18±0.37**	**67.57±0.14**	**64.28±0.14**
SIGN(3,1,1)	**72.24±0.32**	**67.76±0.09**	**64.39±0.18**
SIGN(3,3,3)	**73.94±0.72**	**68.6±0.04**	**65.11±0.14**

[4], paths [18], or motifs [44] that can be tailored to the specific problem. Furthermore, temporal information can be integrated e.g. in the form of temporal motifs [46].

Limitations While our method relies on linear graph aggregation operations of the form $\mathbf{BX}$ for efficient precomputation, it is possible to make the diffusion operator dependent on the node features (and edge features, if available). In particular, graph attention [57] and similar mechanisms [42] use $\mathbf{B}_{\boldsymbol{\theta}}(\mathbf{X})$, where $\boldsymbol{\theta}$ are learnable parameters. The limitation is that such operators preclude efficient precomputation, which is key to the efficiency of our approach. Attention can be implemented in our scheme by training on a small subset of the graph to first determine the attention parameters, then fixing them to precompute the diffusion operator that is used during training and inference.

References

[1] 2017. Wikipedia links, English network dataset – KONECT. `http://konect.uni-koblenz.de/networks/wikipedia_link_en`

[2] Uri Alon and Eran Yahav. 2020. On the Bottleneck of Graph Neural Networks and its Practical Implications. arXiv:cs.LG/2006.05205

[3] James Atwood and Don Towsley. 2016. Diffusion-Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*. 1993–2001.

[4] Sergio Barbarossa and Stefania Sardellitti. 2019. Topological Signal Processing over Simplicial Complexes. *arXiv:1907.11577* (2019).

[5] Peter W Battaglia et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261* (2018).

[6] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2546–2554. `http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`

[7] K. Bhatia, K. Dahiya, H. Jain, A. Mittal, Y. Prabhu, and M. Varma. 2016. The extreme classification repository: Multi-label datasets and code. `http://manikvarma.org/downloads/XC/XMLRepository.html`

[8] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. 2019. Mincut pooling in graph neural networks. *arXiv:1907.00481* (2019).

[9] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Proc. Magazine* 34, 4 (July 2017), 18–42.

[10] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *ICLR*.

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

[12] Jianfei Chen and Jun Zhu. 2018. Stochastic Training of Graph Convolutional Networks.

[13] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *KDD*.

[14] Nicholas Choma, Federico Monti, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Michael Bronstein, Spencer Klein, and Joan Bruna. 2018. Graph neural networks for icecube signal classification. In *ICMLA*.

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.

[16] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. 2007. Weighted graph cuts without eigenvectors a multilevel approach. *PAMI* 29, 11 (2007), 1944–1957.

[17] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarelli, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NIPS*.

[18] Daniel Flam-Shepherd, Tony Wu, Pascal Friederich, and Alan Aspuru-Guzik. 2020. Neural Message Passing on High Order Paths. *arXiv:2002.10413* (2020).

[19] Pablo Gainza et al. 2019. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods* 17 (2019), 184–192.

[20] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*.

[21] Shunwang Gong, Mehdi Bahri, Michael M Bronstein, and Stefanos Zafeiriou. 2020. Geometrically Principled Connections in Graph Neural Networks. In *Proc. CVPR*.

[22] Mark Granovetter. 1982. The Strength of Weak Ties: A Network Theory Revisited. In *Sociological Theory*. 105–130.

[23] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*.

[24] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[25] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin* (2017).

[26] Taher H Haveliwala. 2003. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering* 15, 4 (2003), 784–796.

[27] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 1026–1034.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. CVPR*.

[29] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *CoRR* abs/2005.00687 (2020). arXiv:2005.00687 `https://arxiv.org/abs/2005.00687v4`

[30] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. In *NIPS*.

[31] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 448–456. `http://proceedings.mlr.press/v37/ioffe15.html`

[32] Anees Kazi et al. 2019. InceptionGCN: Receptive Field Aware Graph Convolutional Network for Disease Prediction. In *Information Processing in Medical Imaging*.

[33] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (12 2014).

[34] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

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

[36] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

[37] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1, 4 (1989), 541–551.

[38] Ron Levie, Michael M Bronstein, and Gitta Kutyniok. 2019. Transferability of Spectral Graph Convolutional Neural Networks. *arXiv:1907.12972* (2019).

[39] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. 2018. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *Trans. Signal Proc.* 67, 1 (2018), 97–109.

[40] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proc. ICCV*.

[41] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive graph convolutional neural networks. In *Proc. AAAI*.

[42] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. 2016. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *CVPR*.

[43] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M. Bronstein. 2019. Fake News Detection on Social Media using Geometric Deep Learning. *CoRR* abs/1902.06673 (2019). arXiv:1902.06673 `http://arxiv.org/abs/1902.06673`

[44] Federico Monti, Karl Otness, and Michael M Bronstein. 2018. Motifnet: a motif-based graph convolutional network for directed graphs. In *DSW*.

[45] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning, ICML (JMLR Workshop and Conference Proceedings)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. JMLR.org, 2014–2023.

[46] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *Proc. Web Search and Data Mining*.

[47] Sarah Parisot, Sofia Ira Ktena, Enzo Ferrante, Matthew Lee, Ricardo Guerrero, Ben Glocker, and Daniel Rueckert. 2018. Disease prediction using graph convolutional networks: Application to Autism Spectrum Disorder and Alzheimer's disease. *Med Image Anal* 48 (2018), 117–130.

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`

[49] Siyuan Qi, Wenguan Wang, Baoxiong Jia, Jianbing Shen, and Song-Chun Zhu. 2018. Learning human-object interactions by graph parsing neural networks. In *ECCV*. 401–417.

[50] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *Proc. ICLR*.

[51] Emanuele Rossi, Federico Monti, Michael Bronstein, and Pietro Liò. 2019. ncRNA Classification with Graph Convolutional Networks. In *KDD Workshop on Deep Learning on Graphs*.

[52] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *Trans. Neural Networks* 20, 1 (2008), 61–80.

[53] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop* (2018).

[54] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE Computer Society, 29–38.

[55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958.

[56] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *CVPR*.

[57] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[58] Kirill Veselkov et al. 2019. HyperFoods: Machine intelligent mapping of cancer-beating molecules in foods. *Scientific Reports* 9, 1 (2019), 1–12.

[59] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*.

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

[61] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks.

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

[63] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*.

[64] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. *arXiv:1907.04931* (2019).

[65] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *NIPS*.

[66] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep learning on graphs: A survey. *arXiv:1812.04202* (2018).

[67] Lingxiao Zhao and Leman Akoglu. 2020. PairNorm: Tackling Oversmoothing in GNNs. In *Proc. ICLR*.

[68] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34, 13 (2018), i457–i466.

[69] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, 14 (Jul 2017), i190–i198. `https://doi.org/10.1093/bioinformatics/btx252`

[70] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. In *Advances in Neural Information Processing Systems*.

A Datasets

A.1 Inductive datasets

`Reddit` [24] and `Flickr` [64] are multiclass classification problems, `Yelp` [64] and `PPI` [69] are multilabel classification instances. In `Reddit`, the task is to predict communities of online posts based on user comments. In `Flickr` the task is image categorization based on the description and common properties of online images. In `Yelp` the objective is to predict business attributes based on customer reviews; the task of `PPI` consists in predicting protein functions from the interactions of human tissue proteins. Further details on the generation of the `Yelp` and `Flickr` datasets can be found in [64].

A.2 Transductive dataset

`ogbn-products` [29] represents an Amazon product co-purchasing network [7] where the task is to predict the category of a product in a multi-class classification setup. Dataset splitting is not random, sales ranking (popularity) is instead used to split nodes into training/validation/test. Top 10% products in the ranking are assigned to the training set, next top 2% to validation and the remaining 88% of products are for testing.

`ogbn-papers100M` [29] represents a directed citation network of $\sim$ 111 million academic papers, where the task is to leverage information from the entire citation network to infer the labels (subject areas) of a smaller subset of ArXiv papers. The splitting strategy is time-based. Specifically, the training nodes (with labels) are all ArXiv papers published until 2017, validation nodes are ArXiv papers published in 2018 and test nodes are ArXiv papers published since 2019.

A.3 Wikipedia

`Wikipedia` links is a large-scale directed network of links between articles in the English version of Wikipedia. For the sake of our timing experiments the network has been turned into undirected. Node features have been randomly generated with a dimensionality of 100 as in `ogbn-products`.

B Model Selection and Hyperparameter Tuning

Tuning involved the following architectural and optimization hyperparameters: weight decay, dropout rate, batch size, learning rate, number of feedforward layers and units both in inception and classification modules. For each inductive experiment we chose the set of hyperparameters matching the best average validation loss calculated over 5 runs. For the the transductive setting we kept, instead, the set of hyperparameters with minimum validation loss over a single run. The hyperparameter search space for the inductive setting and grid for the transductive one are described in Table 9. The estimated hyperparameters for each best *SIGN* configuration are reported in Table 10 for inductive datasets and Table 11 for the transductive ones.

C Triangle-based Operators

The triangle operator encodes the concept of *homophily* with a stronger acceptation with respect to the adjacency matrix: two nodes are connected by an edge only if they are both part of the same closed triad, i.e. if they are connected together and are both connected to the same node. Edge weights are proportional to the amount of triangles an edge belongs to, and they are normalised row-wise so to represent, for each node in a neighbourhood, its relative importance with respect to all the other neighbors.

This brings us to two considerations: first of all, the triangle operator is not carrying information related to nodes which were not already in the neighborhood. Secondly, it emphasizes the connections with those neighbors which are more related to our source node in virtue of the relationship described above. We can thus envision this operator being more useful in those graphs where this kind of relationship can be more discriminative within a neigborhood.

To verify this, in Figure 3 we plot the normalized frequency distribution of intra-neighborhood standard deviation for the weights of triangle operators. It is interesting to notice the significantly

Table 9: Hyperparameter search space/grid. Ranges in the form [*low*, *high*] and sampling distributions. *Inception Layers* and *Classification Layers* are the number of feedforward layers in the representation part of the model (replacing Θ) and the classification part of the model (replacing Ω) respectively. The only exception is represented by *Yelp*, for which the Ω module was kept shallow (no hidden layers) to allow for lighter training and the left bounds on the dropout, learning rate and batch size intervals were lowered to, respectively, 0.0, 0.00001 and 60.

	TRANSDUCTIVE	INDUCTIVE	
HYPERPARAMETER	VALUES	SPACE	DISTRIBUTION
Learning Rate	0.0001, 0.001	[0.0001, 0.0025]	UNIFORM
Batch Size	4096, 8192, 16384	[128, 2048]	QUANTIZED UNIFORM
Dropout	0.5	[0.2, 0.8]	UNIFORM
Weight Decay	0.0, 0.00001	[0, 0.0001]	UNIFORM
Inception Layers	1	1, 2	—
Inception Units	256, 512	[128, 512]	QUANTIZED UNIFORM
Classification Layers	1	1, 2	—
Classification Units	256, 512	[512, 1024]	QUANTIZED UNIFORM
Activation	PRELU	RELU, PRELU	—

Table 10: Hyperparameters chosen for the best configuration of SIGN on inductive datasets.

HYPERPARAMETER	REDDIT	FLICKR	PPI	YELP
Learning Rate	0.00012278578238312588	0.0017230142114465549	0.0014386686616183625	0.00005
Dropout	0.707328910934901	0.7608352140584778	0.3085607444207686	0.05
Weight Decay	9.176773905054599E-05	9.419820474221673E-05	3.2571631135664696E-06	4.452466189193362E-07
Batch Size	830	330	210	90
Inception Layers	1	2	2	2
Inception Units	460	465	315	320
Classification Layers	1	1	2	0
Classification Units	675	925	870	—
Activation	RELU	PRELU	RELU	RELU

Table 11: Hyperparameters chosen for the best configuration of SIGN on `ogbn-product` and those used on the `ogbn-product` dataset.

HYPERPARAMETER	OGBN-PRODUCTS	OGBN-PAPERS100M
Learning Rate	0.0001	0.001
Dropout	0.5	0.1
Weight Decay	0.0001	0.0
Batch Size	4096	256
Inception Layers	1	1
Inception Units	512	256
Classification Layers	1	3
Classification Units	512	256
Activation	PRELU	RELU

different trends characterizing `Flickr` and `Reddit`, two datasets where triangle operators have experimentally brought, respectively, relative large and small performance improvement. `Flickr` tends to exhibit larger weight variations than other datasets, while, on the contrary, `Reddit` is the dataset where the smallest intra-neighborhood variation is observed. This suggests how, in `Flickr`, the triangle operator is able to restrict feature aggregation to a subset of the original neighbors –those co-occurring in the larger number of triangles– while in `Reddit` it mostly boils down to uniform averaging, making this operator not much more expressive than a simple adjacency matrix.

For replicability we report that, in the computation of triangle operators for `PPI`, we retained the self-loops already present in the original dataset. Investigations on how the presence of these edges affects the expressiveness of the triangle operator are left for future work.

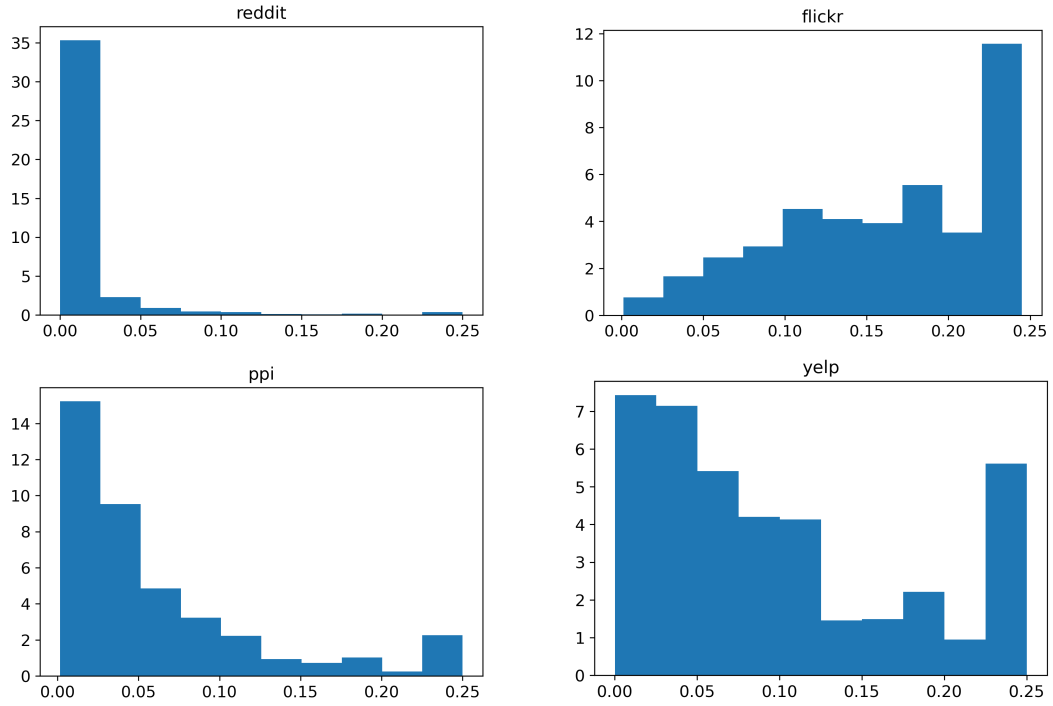

Figure 3: Normalized frequency distributions for row-wise variations on the diffusion weights of triangle operators over inductive datasets. Variations are measured as the standard deviation on the weight value over original neighborhoods from the test graph.