FAST GRAPH REPRESENTATION LEARNING WITH PYTORCH GEOMETRIC

Matthias Fey & Jan E. Lenssen
Department of Computer Graphics
TU Dortmund University
44227 Dortmund, Germany
{matthias.fey,janeric.lenssen}@udo.edu

ABSTRACT

We introduce *PyTorch Geometric*, a library for deep learning on irregularly structured input data such as graphs, point clouds and manifolds, built upon PyTorch. In addition to general graph data structures and processing methods, it contains a variety of recently published methods from the domains of relational learning and 3D data processing. PyTorch Geometric achieves high data throughput by leveraging sparse GPU acceleration, by providing dedicated CUDA kernels and by introducing efficient mini-batch handling for input examples of different size. In this work, we present the library in detail and perform a comprehensive comparative study of the implemented methods in homogeneous evaluation scenarios.

1 INTRODUCTION

Graph Neural Networks (GNNs) recently emerged as a powerful approach for representation learning on graphs, point clouds and manifolds (Bronstein et al., 2017; Kipf & Welling, 2017). Similar to the concepts of convolutional and pooling layers on regular domains, GNNs are able to (hierarchically) extract localized embeddings by passing, transforming, and aggregating information (Bronstein et al., 2017; Gilmer et al., 2017; Battaglia et al., 2018; Ying et al., 2018; Morris et al., 2019).

However, implementing GNNs is challenging, as high GPU throughput needs to be achieved on highly sparse and irregular data of varying size. Here, we introduce *PyTorch Geometric* (PyG), a geometric deep learning extension library for PyTorch (Paszke et al., 2017) which achieves high performance by leveraging dedicated CUDA kernels. Following a simple message passing API, it bundles most of the recently proposed convolutional and pooling layers into a single and unified framework. All implemented methods support both CPU and GPU computations and follow an immutable data flow paradigm that enables dynamic changes in graph structures through time. PyG is released under the MIT license and is available on GitHub.[1] It is thoroughly documented and provides accompanying tutorials and examples as a first starting point.[2]

2 OVERVIEW

In PyG, we represent a graph $\mathcal{G} = (\boldsymbol{X}, (\boldsymbol{I}, \boldsymbol{E}))$ by a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ of N nodes and a sparse adjacency tuple $(\boldsymbol{I}, \boldsymbol{E})$ of E edges, where $\boldsymbol{I} \in \mathbb{N}^{2 \times E}$ encodes edge indices in COOrdinate (COO) format and $\boldsymbol{E} \in \mathbb{R}^{E \times D}$ (optionally) holds D-dimensional edge features. All user facing APIs, *e.g.*, data loading routines, multi-GPU support, data augmentation or model instantiations are heavily inspired by PyTorch to keep them as familiar as possible.

Neighborhood Aggregation. Generalizing the convolutional operator to irregular domains is typically expressed as a *neighborhood aggregation* or *message passing* scheme (Gilmer et al., 2017)

$$\vec{x}_i' = \gamma \left(\vec{x}_i, \square_{j \in \mathcal{N}(i)} \phi \left(\vec{x}_i, \vec{x}_j, \vec{e}_{j,i} \right) \right) \tag{1}$$

[1] GitHub repository: https://github.com/rusty1s/pytorch_geometric
[2] Documentation: https://rusty1s.github.io/pytorch_geometric

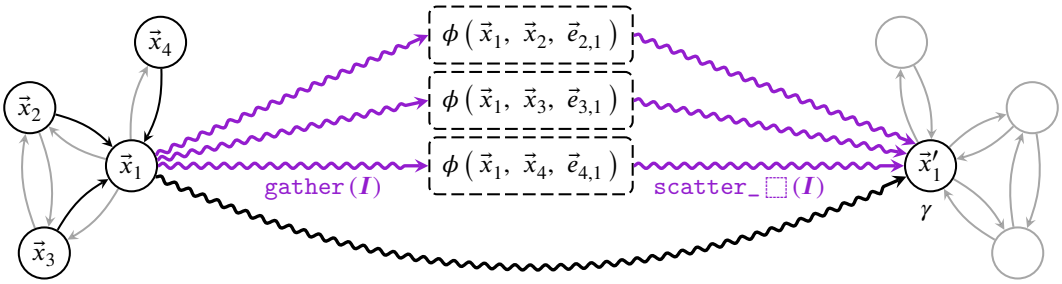

Figure 1: Computation scheme of a GNN layer by leveraging gather and scatter methods based on edge indices $\boldsymbol{I}$, hence alternating between node parallel space and edge parallel space.

where ⬚ denotes a differentiable, permutation invariant function, *e.g.*, sum, mean or max, and γ and ϕ denote differentiable functions, *e.g.*, MLPs. In practice, this can be achieved by gathering and scattering of node features and vectorized element-wise computation of γ and ϕ, as visualized in Figure 1. Although working on irregularly structured input, this scheme can be heavily accelerated by the GPU. In contrast to implementations via sparse matrix multiplications, the usage of gather/scatter proves to be advantageous for low-degree graphs and non-coalesced input (*cf.* Appendix A), and allows for the integration of central node and multi-dimensional edge information while aggregating.

We provide the user with a general `MessagePassing` interface to allow for rapid and clean prototyping of new research ideas. In order to use, users only need to define the methods ϕ, *i.e.*, `message`, and γ, *i.e.*, `update`, as well as chosing an aggregation scheme ⬚. For implementing ϕ, node features are automatically mapped to the respective source and target nodes.

Almost all recently proposed neighborhood aggregation functions can be lifted to this interface, including (but not limited to) the methods already integrated into PyG: For learning on arbitrary graphs we have implemented GCN (Kipf & Welling, 2017) and its simplified version (SGC) from Wu et al. (2019), the spectral chebyshev and ARMA filter convolutions (Defferrard et al., 2016; Bianchi et al., 2019), GraphSAGE (Hamilton et al., 2017), the attention-based operators GAT (Veličković et al., 2018) and AGNN (Thekumparampil et al., 2018), the Graph Isomorphism Network (GIN) from Xu et al. (2019), the Approximate Personalized Propagation of Neural Predictions (APPNP) operator (Klicpera et al., 2019), the Dynamic Neighborhood Aggregation (DNA) operator (Fey, 2019) and the signed operator for learning in signed networks (Derr et al., 2018).

For learning on point clouds, manifolds and graphs with multi-dimensional edge features, we provide the relational GCN operator from Schlichtkrull et al. (2018), PointNet++ (Qi et al., 2017), PointCNN (Li et al., 2018), and the continuous kernel-based methods MPNN (Gilmer et al., 2017; Simonovsky & Komodakis, 2017), MoNet (Monti et al., 2017), SplineCNN (Fey et al., 2018) and the edge convolution operator (EdgeCNN) from Wang et al. (2018b).

In addition to these operators, we provide high-level implementations of, *e.g.*, maximizing mutual information (Veličković et al., 2019), autoencoding graphs (Kipf & Welling, 2016; Pan et al., 2018), aggregating jumping knowledge (Xu et al., 2018), and predicting temporal events in knowledge graphs (Jin et al., 2019).

Global Pooling. PyG also supports graph-level outputs as opposed to node-level outputs by providing a variety of *readout* functions such as global add, mean or max pooling. We additionaly offer more sophisticated methods such as set-to-set (Vinyals et al., 2016), sort pooling (Zhang et al., 2018) or the global soft attention layer from Li et al. (2016).

Hierarchical Pooling. To further extract hierarchical information and to allow deeper GNN models, various pooling approaches can be applied in a spatial or data-dependent manner. We currently provide implementation examples for Graclus (Dhillon et al., 2007; Fagginger Auer & Bisseling, 2011) and voxel grid pooling (Simonovsky & Komodakis, 2017), the iterative farthest point sampling algorithm (Qi et al., 2017) followed by k-NN or query ball graph generation (Qi et al., 2017; Wang et al., 2018b), and differentiable pooling mechanisms such as DiffPool (Ying et al., 2018) and top_k pooling (Gao & Ji, 2018; Cangea et al., 2018).

Table 1: Semi-supervised node classification with both fixed and random splits.

Method	Cora		CiteSeer		PubMed	
	Fixed	Random	Fixed	Random	Fixed	Random
Cheby	81.4 ± 0.7	77.8 ± 2.2	70.2 ± 1.0	67.7 ± 1.7	78.4 ± 0.4	75.8 ± 2.2
GCN	81.5 ± 0.6	79.4 ± 1.9	71.1 ± 0.7	68.1 ± 1.7	79.0 ± 0.6	77.4 ± 2.4
GAT	83.1 ± 0.4	81.0 ± 1.4	70.8 ± 0.5	69.2 ± 1.9	78.5 ± 0.3	78.3 ± 2.3
SGC	81.7 ± 0.1	80.2 ± 1.6	71.3 ± 0.2	68.7 ± 1.6	78.9 ± 0.1	76.5 ± 2.4
ARMA	82.8 ± 0.6	80.7 ± 1.4	$\mathbf{72.3 \pm 1.1}$	68.9 ± 1.6	78.8 ± 0.3	77.7 ± 2.6
APPNP	$\mathbf{83.3 \pm 0.5}$	$\mathbf{82.2 \pm 1.5}$	71.8 ± 0.5	$\mathbf{70.0 \pm 1.4}$	$\mathbf{80.1 \pm 0.2}$	$\mathbf{79.4 \pm 2.2}$

Mini-batch Handling. Our framework supports batches of multiple graph instances (of potentially different size) by automatically creating a single (sparse) block-diagonal adjacency matrix and concatenating feature matrices in the node dimension. Therefore, neighborhood aggregation methods can be applied without modification, since no messages are exchanged between disconnected graphs. In addition, an automatically generated assignment vector ensures that node-level information is not aggregated across graphs, *e.g.*, when executing global aggregation operators.

Processing of Datasets. We provide a consistent data format and an easy-to-use interface for the creation and processing of datasets, both for large datasets and for datasets that can be kept in memory during training. In order to create new datasets, users just need to read/download their data and convert it to the PyG data format in the respective `process` method. In addition, datasets can be modified by the use of `transforms`, which take in separate graphs and transform them, *e.g.*, for data augmentation, for enhancing node features with synthetic structural graph properties (Cai & Wang, 2018), to automatically generate graphs from point clouds or to sample point clouds from meshes.

PyG already supports a lot of common benchmark datasets often found in literature which are automatically downloaded and processed on first instantiation. In detail, we provide over 60 graph kernel benchmark datasets[3] (Kersting et al., 2016), *e.g.*, PROTEINS or IMDB-BINARY, the citation graphs Cora, CiteSeer, PubMed and Cora-Full (Sen et al., 2008; Bojchevski & Günnemann, 2018), the Coauthor CS/Physics and Amazon Computers/Photo datasets from Shchur et al. (2018), the molecule datasets QM7b (Montavon et al., 2013) and QM9 (Ramakrishnan et al., 2014), the protein-protein interaction graphs from Hamilton et al. (2017), and the temporal datasets Bitcoin-OTC (Kumar et al., 2016), ICEWS (Boschee et al., 2015) and GDELT (Leetaru & Schrodt, 2013). In addition, we provide embedded datasets like MNIST superpixels (Monti et al., 2017), FAUST (Bogo et al., 2014), ModelNet10/40 (Wu et al., 2015), ShapeNet (Chang et al., 2015), COMA (Ranjan et al., 2018) and the PCPNet dataset from Guerrero et al. (2018).

3 Empirical Evaluation

We evaluate the correctness of the implemented methods by performing a comprehensive comparative study in homogeneous evaluation scenarios. Descriptions and statistics of all used datasets can be found in Appendix B. For all experiments, we tried to follow the hyperparameter setup of the respective papers as closely as possible. The individual experimental setups can be derived and all experiments can be replicated from the code provided at our GitHub repository.[4]

Semi-supervised Node Classification. We perform semi-supervised node classification (*cf.* Table 1) by reporting average accuracies of (a) 100 runs for the fixed train/val/test split from Kipf & Welling (2017), and (b) 100 runs of randomly initialized train/val/test splits as suggested by Shchur et al. (2018), where we additionally ensure uniform class distribution on the train split.

Nearly all experiments show a high reproducibility of the results reported in the respective papers. However, test performance is worse for all models when using random data splits. Among the experiments, the APPNP operator (Klicpera et al., 2019) generally performs best, while the ARMA

[3]Kernel datasets: `http://graphkernels.cs.tu-dortmund.de`

[4]`https://github.com/rusty1s/pytorch_geometric/benchmark`

Table 2: Graph classification.

	Method	MUTAG	PROTEINS	COLLAB	IMDB-BINARY	REDDIT-BINARY
Flat	GCN	74.6 ± 7.7	73.1 ± 3.8	$\mathbf{80.6 \pm 2.1}$	72.6 ± 4.5	89.3 ± 3.3
	SAGE	74.9 ± 8.7	$\mathbf{73.8 \pm 3.6}$	79.7 ± 1.7	72.4 ± 3.6	89.1 ± 1.9
	GIN-0	$\mathbf{85.7 \pm 7.7}$	72.1 ± 5.1	79.3 ± 2.7	$\mathbf{72.8 \pm 4.5}$	89.6 ± 2.6
	GIN-ϵ	83.4 ± 7.5	72.6 ± 4.9	79.8 ± 2.4	72.1 ± 5.1	$\mathbf{90.3 \pm 3.0}$
Hier.	Graclus	77.1 ± 7.2	73.0 ± 4.1	79.6 ± 2.0	72.2 ± 4.2	88.8 ± 3.2
	top_k	76.3 ± 7.5	72.7 ± 4.1	$\mathbf{79.7 \pm 2.2}$	72.5 ± 4.6	87.6 ± 2.4
	DiffPool	$\mathbf{85.0 \pm 10.3}$	$\mathbf{75.1 \pm 3.5}$	78.9 ± 2.3	$\mathbf{72.6 \pm 3.9}$	$\mathbf{92.1 \pm 2.6}$
Global	SAGE w/o JK	73.7 ± 7.8	72.7 ± 3.6	79.6 ± 2.4	72.1 ± 4.4	87.9 ± 1.9
	GlobalAttention	74.6 ± 8.0	72.5 ± 4.5	$\mathbf{79.6 \pm 2.2}$	72.3 ± 3.8	87.4 ± 2.5
	Set2Set	73.7 ± 6.9	$\mathbf{73.6 \pm 3.7}$	79.6 ± 2.3	72.2 ± 4.2	$\mathbf{89.6 \pm 2.4}$
	SortPool	$\mathbf{77.3 \pm 8.9}$	72.4 ± 4.1	77.7 ± 3.1	$\mathbf{72.4 \pm 3.8}$	74.9 ± 6.7

Table 3: Point cloud classification.

Method	ModelNet10
MPNN	92.07
PointNet++	92.51
EdgeCNN	92.62
SplineCNN	92.65
PointCNN	**93.28**

Table 4: Training runtime comparison.

Dataset	Method	DGL DB	DGL GS	PyG
Cora	GCN	4.19s	0.32s	**0.25s**
	GAT	6.31s	5.36s	**0.80s**
CiteSeer	GCN	3.78s	0.34s	**0.30s**
	GAT	5.61s	4.91s	**0.88s**
PubMed	GCN	12.91s	0.36s	**0.32s**
	GAT	18.69s	13.76s	**2.42s**
MUTAG	RGCN	18.81s	2.40s	**2.14s**

(Bianchi et al., 2019), SGC (Wu et al., 2019), GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018) operators follow closely behind.

Graph Classification. We report the average accuracy of 10-fold cross validation on a number of common benchmark datasets (*cf.* Table 2) where we randomly sample a training fold to serve as a validation set. We only make use of discrete node features. In case they are not given, we use one-hot encodings of node degrees as feature input. For all experiments, we use the global mean operator to obtain graph-level outputs. Inspired by the Jumping Knowledge framework (Xu et al., 2018), we compute graph-level outputs after each convolutional layer and combine them via concatenation. For evaluating the (global) pooling operators, we use the GraphSAGE operator as our baseline. We omit Jumping Knowledge when comparing global pooling operators, and hence report an additional baseline based on global mean pooling. For each dataset, we tune (1) the number of hidden units $\in \{16, 32, 64, 128\}$ and (2) the number of layers $\in \{2, 3, 4, 5\}$ with respect to the validation set.

Due to standardized evaluations and network architectures, not all results are aligned with their official reported values. For example, except for DiffPool (Ying et al., 2018), (global) pooling operators do not perform as benefically as expected to their respective (flat) counterparts, especially when baselines are enhanced by Jumping Knowledge (Xu et al., 2018). However, the potential of more sophisticated approaches may not be well-reflected on these simple benchmark tasks (Cai & Wang, 2018). Among the flat GNN approaches, the GIN layer (Xu et al., 2019) generally achieves the best results.

Point Cloud Classification. We evaluate various point cloud methods on ModelNet10 (Wu et al., 2015) where we uniformly sample 1,024 points from mesh surfaces based on face area (*cf.* Table 3). As hierarchical pooling layers, we use the iterative farthest point sampling algorithm followed by a new graph generation based on a larger query ball (PointNet++ (Qi et al., 2017), MPNN (Gilmer et al., 2017; Simonovsky & Komodakis, 2017) and SplineCNN (Fey et al., 2018)) or based on a fixed

number of nearest neighbors (EdgeCNN (Wang et al., 2018b) and PointCNN (Li et al., 2018)). We have taken care to use approximately the same number of parameters for each model.

All approaches perform nearly identically with PointCNN (Li et al., 2018) taking a slight lead. We attribute this to the fact that all operators are based on similar principles and might have the same expressive power for the given task.

Runtime Experiments. We conduct several experiments on a number of dataset-model pairs to report the runtime of a whole training procedure for 200 epochs obtained on a single NVIDIA GTX 1080 Ti (*cf.* Table 4). As it shows, PyG is very fast despite working on sparse data. Compared to the *Degree Bucketing* (DB) approach of the *Deep Graph Library* (DGL) v0.2 (Wang et al., 2018a), PyG trains models up to 40 times faster. Although runtimes are comparable when using gather and scatter optimizations (GS) inside DGL, we could further improve runtimes of GAT (Veličković et al., 2018) by up to 7 times by providing our own optimized sparse softmax kernels.

4 Roadmap and Conclusion

We presented the PyTorch Geometric framework for fast representation learning on graphs, point clouds and manifolds. We are actively working to further integrate existing methods and plan to quickly integrate future methods into our framework. All researchers and software engineers are invited to collaborate with us in extending its scope.

Acknowledgments

This work has been supported by the *German Research Association (DFG)* within the Collaborative Research Center SFB 876, *Providing Information by Resource-Constrained Analysis*, projects A6 and B2. We thank Moritz Ludolph and all other contributors for their amazing involvement in this project. Last but not least, we thank Christopher Morris for fruitful discussions, proofreading and helpful advice.

References

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Graph neural networks with convolutional ARMA filters. *CoRR*, abs/1901.01343, 2019.

F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *CVPR*, 2014.

A. Bojchevski and S. Günnemann. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. In *ICLR*, 2018.

E. Boschee, J. Lautenschlager, S. O'Brien, S. Shellman, J. Starz, and M. Ward. ICEWS coded event data. *Harvard Dataverse*, 2015.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. In *Signal Processing Magazine*, 2017.

C. Cai and Y. Wang. A simple yet effective baseline for non-attribute graph classification. *CoRR*, abs/1811.03508, 2018.

C. Cangea, P. Veličković, N. Jovanović, T. N. Kipf, and P. Liò. Towards sparse hierarchical graph classifiers. In *NeurIPS-W*, 2018.

A. X. Chang, T. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. *CoRR*, abs/1512.03012, 2015.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

T. Derr, Y. Ma, and J. Tang. Signed graph convolutional networks. In *ICDM*, 2018.

I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. In *TPAMI*, 2007.

B. O. Fagginger Auer and R. H. Bisseling. A GPU algorithm for greedy graph matching. In *Facing the Multicore - Challenge II - Aspects of New Paradigms and Technologies in Parallel Computing*, 2011.

M. Fey. Just jump: Dynamic neighborhood aggregation in graph neural networks. In *ICLR-W*, 2019.

M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *CVPR*, 2018.

H. Gao and S. Ji. Graph U-Net. `https://openreview.net/forum?id=HJePRoAct7`, 2018. Submitted to ICLR.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37, 2018.

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

W. Jin, C. Zhang, P. Szekely, and X. Ren. Recurrent event network for reasoning over temporal knowledge graphs. In *ICLR-W*, 2019.

K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. Benchmark data sets for graph kernels. `http://graphkernels.cs.tu-dortmund.de`, 2016.

T. N. Kipf and M. Welling. Variational graph auto-encoders. In *NIPS-W*, 2016.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized PageRank. In *ICLR*, 2019.

S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos. Edge weight prediction in weighted signed networks. In *ICDM*, 2016.

K. Leetaru and P. A. Schrodt. GDELT: Global data on events, location, and tone. *ISA Annual Convention*, 2013.

Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. PointCNN: Convolution on $\mathcal{X}$-transformed points. In *NeurIPS*, 2018.

G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K. Müller, and O. A. von Lilienfeld. Machine learning of molecular electronic properties in chemical compound space. *New Journal of Physics*, 2013.

F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*, 2017.

C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. Adversarially regularized graph autoencoder for graph embedding. In *IJCAI*, 2018.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 2014.

A. Ranjan, T. Bolkart, S. Sanyal, and M. J. Black. Generating 3D faces using convolutional mesh autoencoders. In *ECCV*, 2018.

M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018.

G. Sen, G. Namata, M. Bilgic, and L. Getoor. Collective classification in network data. *AI Magazine*, 29, 2008.

O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. In *NeurIPS-W*, 2018.

M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*, 2017.

K. K. Thekumparampil, C. Wang, S. Oh, and L. Li. Attention-based graph neural network for semi-supervised learning. *CoRR*, abs/1803.03735, 2018.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjeml. Deep graph infomax. In *ICLR*, 2019.

O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

M. Wang, L. Yu, A. Gan, D. Zheng, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, J. Zhao, H. Lin, C. Ma, D. Deng, Q. Guo, H. Zhang, J. Li, A. J. Smola, and Z. Zhang. Deep graph library. `http://dgl.ai`, 2018a.

Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018b.

F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. *CoRR*, abs/1902.07153, 2019.

Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015.

K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

R. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

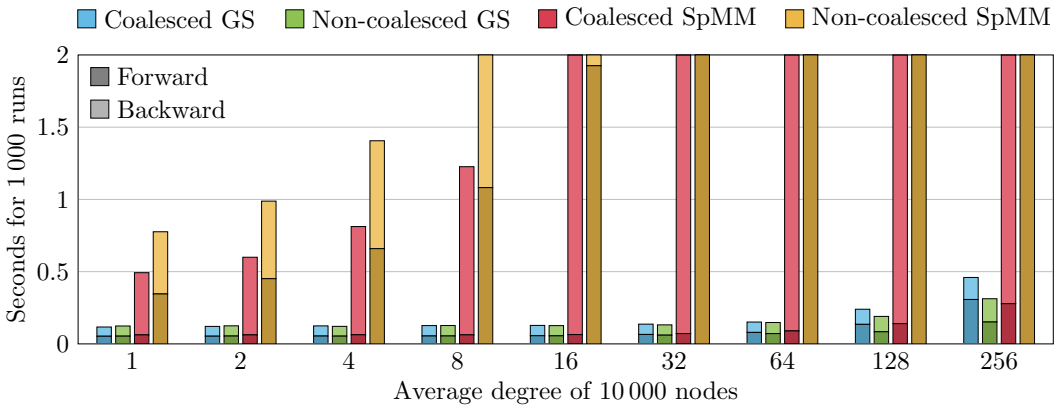

Figure 2: Forward and backward runtimes of 1 000 runs of gather and scatter operations (GS) in comparison to sparse-matrix multiplications (SpMM) on Erdős RÃľnyi graphs with 10 000 nodes and varying average node degrees. Runtimes are capped at two seconds due to visualization. We report runtimes for both coalesced (*i.e.* ordered by row index) and non-coalesced sparse layout.

A Gather and Scatter Operations

PyTorch Geometric makes heavy usage of gather and scatter operations to map node and edge information into edge and node parallel space, respectively. Despite inevitable non-coalesced memory access, these operations achieve high data throughput by leveraging parallelization over *all* elements and making use of atomic operations with approximately constant runtime in practice. Following upon the PyTorch `scatter_add` implementation, we provide our own `mean` and `max` operations to allow for all different kinds of aggregation.[5]

Figure 2 compares the runtime of gather and scatter operations (GS) to the frequently used alternative of using sparse-matrix multiplications (SpMM). It shows that atomic operations only begin to throttle the runtime when scattering graphs with high average node degree ($\approx$ 128) and even exceed the runtime of highly optimized SpMM executions, both for forward and backward passes.

Due to SpMM first converting adjacency matrices into *Compressed Row Storage* (CSR) format, it expects *coalesced* sparse tensors (*i.e.* ordered by row index) which is expensive to compute on GPUs and should be hence performed as part of the pre-processing. However, for the backward pass of SpMM, coalescing is performed in any case due to the need of transposing the sparse tensor. In contrast, GS is always fast, nevertheless of the input being coalesced. Additionally, it allows for modifications of the graph connectivity (*i.e.* adding self-loops), allows bidirectional data flow, and does naturally support the integration of central node and multi-dimensional edge information.

However, we do think that our GS scheme can still be improved, *e.g.*, in highly dense graph settings and towards reducing the memory footprint in the edge parallel space. In addition, it should be noted that scatter operations are non-deterministic by nature on the GPU. Although we did not observe any deviations for inference, training results can vary across the same manual seeds.

B Datasets

We give detailed descriptions and statistics (*cf.* Table 5) of the datasets used in our experiments:

Citation Networks. In the citation network datasets Cora, Citeseer and Pubmed nodes represent documents and edges represent citation links. The networks contain bag-of-words feature vectors for each document. We treat the citation links as (undirected) edges. For training, we use 20 labels per class.

[5]GitHub repository: `https://github.com/rusty1s/pytorch_scatter`

Table 5: Statistics of the datasets used in the experiments.

Dataset	Graphs	Nodes	Edges	Features	Classes	Label rate
Cora	1	2,708	5,278	1,433	7	0.052
CiteSeer	1	3,327	4,552	3,703	6	0.036
PubMed	1	19,717	44,324	500	3	0.003
MUTAG	188	17.93	19.79	7	2	0.800
PROTEINS	1,113	39.06	72.82	3	2	0.800
COLLAB	5,000	74.49	2,457.22	—	3	0.800
IMDB-BINARY	1,000	19.77	96.53	—	2	0.800
REDDIT-BINARY	2,00	429.63	497.754	—	2	0.800
ModelNet10	4,899	1,024	∼19,440	—	10	0.815

Social Network Datasets. COLLAB is derived from three public scientific collaboration datasets. Each graph corresponds to an ego-network of different researchers from each field with the task to label each graph to the field the corresponding researcher belongs to. IMDB-BINARY is a movie collaboration dataset where each graph corresponds to an ego-network of actors/actresses. An edge is drawn between two actors/actresses if they appear in the same movie. The task is to infer the genre of the graph. REDDIT-BINARY is an online discussion dataset where each graph corresponds to a thread. An edge is drawn between two users if one of them responded to another's comment. The task is to label each graph to the community/subreddit it belongs to.

Bioinformatic Datasets. MUTAG is a dataset consisting of mutagenetic aromatic and heteroaromatic nitro compounds. PROTEINS holds a set of proteins represented by graphs. Nodes represent secondary structure elements (SSEs) which are connected whenever there are neighbors either in the amino acid sequence or in 3D space.

3D Object Datasets. ModelNet10 is an orientation-aligned dataset of CAD models. Each model corresponds to exactly one out of 10 object categories. Categories were chosen based on a list of the most common object categories in the world.