DEEP GRAPH LIBRARY: TOWARDS EFFICIENT AND SCALABLE DEEP LEARNING ON GRAPHS

Minjie Wang [1 3], Lingfan Yu [1], Da Zheng [3], Quan Gan [4], Yu Gai [2], Zihao Ye [4], Mufei Li [4], Jinjing Zhou [4], Qi Huang [2], Chao Ma [4], Ziyue Huang [5], Qipeng Guo [6], Hao Zhang [7], Haibin Lin [3], Junbo Zhao [1], Jinyang Li [1], Alexander Smola [3], Zheng Zhang [4 2]
[1] New York University, [2] NYU Shanghai, [3] Amazon Web Services, [4] AWS Shanghai AI Lab
[5] Hong Kong University of Science and Technology, [6] Fudan University,
[7] Chongqing University of Posts and Telecommunications

ABSTRACT

Accelerating research in the emerging field of deep graph learning requires new tools. Such systems should support graph as the core abstraction and take care to maintain *both* forward (i.e. supporting new research ideas) and backward (i.e. integration with existing components) compatibility. In this paper, we present Deep Graph Library (DGL). DGL enables arbitrary message handling and mutation operators, flexible propagation rules, and is framework agnostic so as to leverage high-performance tensor, autograd operations, and other feature extraction modules already available in existing frameworks. DGL carefully handles the sparse and irregular graph structure, deals with graphs big and small which may change dynamically, fuses operations, and performs auto-batching, all to take advantages of modern hardware. DGL has been tested on a variety of models, including but not limited to the popular Graph Neural Networks (GNN) and its variants, with promising speed, memory footprint and scalability.

1 INTRODUCTION

Learning from structured data like graphs is widely regarded as an important problem (Wu et al., 2019b), because the graph is inherently a more general form of data structure than tensors. A broad range of models can be unified as either learning from explicit or inferring latent structures. Examples include TreeLSTM (Tai et al., 2015) that works on sentence parsing trees, Capsule Network (Sabour et al., 2017) and Transformer (Vaswani et al., 2017) that learns soft connections among entities (e.g. capsules, words). Recently, Graph neural networks (GNNs) is a rising family of models that aim to model a set of node entities together with their relationships (edges). The application regime of the GNN framework is broad, such as chemical molecules, social networks, knowledge graphs and recommender systems (Zitnik et al., 2018; Schlichtkrull et al., 2018; Hamilton et al., 2018; Ying et al., 2018a).

Unfortunately, existing tensor-based frameworks (e.g. Tensorflow, Pytorch, MXNet) lack intuitive support for this trend of deep graph learning. Specifically, GNNs are defined using the message passing paradigm (Gilmer et al., 2017), which can be seen as a model inductive bias to facilitate information flow across the graph. However, tensor-based frameworks do not support the message-passing interface. As such, researchers need to manually emulate graph computation using tensor operations, which poses an implementation challenge.

During the past year we have seen the release of several graph training systems (Battaglia et al., 2018; Ma et al., 2018; Alibaba, 2019; Fey & Lenssen, 2019). However, most if not all of these libraries compromise programming flexibility to boost the performance of a narrow range of GNNs, as we briefly summarize in the Table 1. As research on deep graph learning is going to evolve and iterate quickly, we dive in to the source of graph learning problems to provide a more comprehensive graph-oriented solution, the **Deep Graph Library (DGL)**[1].

[1] Project Website: `http://dgl.ai`

Currently, with DGL, we provide 1) graph as the central abstraction for users; 2) flexible APIs allowing arbitrary message-passing computation over a graph; 3) support for gigantic and dynamic graphs; 4) efficient memory usage and high training speed.

DGL is **platform-agnostic** so that it can easily be integrated with tensor-oriented frameworks like PyTorch and MXNet. It is an open-source project under active development. Appendix A summarizes the models released in DGL repository. In this paper, we compare DGL against the state-of-the-art library on multiple standard GNN setups and show the improvement of training speed and memory efficiency.

2 Framework requirements of Deep Learning on Graphs

Message passing paradigm. Formally, we define a graph $G(V, E)$. V is the set of nodes with $\mathbf{v}_i$ being the feature vector associated with each node. E is the set of the edge tuples $(\mathbf{e}_k, r_k, s_k)$, where $s_k \to r_k$ represents the edge from node s_k to r_k, and $\mathbf{e}_k$ is feature vector associated with the edge. DGNs are defined by the following edge-wise and node-wise computation:

$$\text{Edge-wise: } \mathbf{m}_k^{(t)} = \phi^e(\mathbf{e}_k^{(t-1)}, \mathbf{v}_{r_k}^{(t-1)}, \mathbf{v}_{s_k}^{(t-1)}), \quad \text{Node-wise: } \mathbf{v}_i^{(t)} = \phi^v(\mathbf{v}_i^{(t-1)}, \bigoplus_{\substack{k \\ \text{s.t. } r_k = i}} \mathbf{m}_k^{(t)}), \tag{1}$$

In Equation 1, ϕ^e is a *message function* defined on each edge to generate a "message" by combining the edge feature with node features at its two ends; ϕ^v is an *update function* defined on each node to update the node feature by aggregating its incoming messages using the *reduce* operation $\oplus$. In GNNs, these functions are parameterized by neural network modules, $\oplus$ can be a sum operation, or mean, max/min, or even an LSTM network (Hamilton et al., 2017). Recent study on the theoretical capability of GNNs (Xu et al., 2018) revealed the limitation of simple reduce operations. Thus, it is crucial for the graph learning system to be flexible and let users express arbitrary ϕ^e, ϕ^v and $\oplus$.

Propagation order. The message passing paradigm defines the computation on each node/edge while the propagation order determines how messages are propagated. One can trigger the computation synchronously (Kipf & Welling, 2017; Veličković et al., 2018; Ying et al., 2018b) (*full propagation*), or following a certain order like TreeLSTM (Tai et al., 2015) and Belief Propagation (Jin et al., 2018) (*partial propagation*), or on some random walks (Perozzi et al., 2014; Grover & Leskovec, 2016) and sampled subgraphs (Hamilton et al., 2017; Chen et al., 2018a; 2017). All said, the propagation aspect is another crucial aspect to consider.

Graph type. One dimension to categorize graphs has to do with scale and instances:

- datasets containing many moderately-sized graph samples such as molecule structures. For this type of dataset, each graph can easily fit into a single GPU, which leaves the main performance challenge to be batching and training across multiple GPUs.
- datasets containing one potentially giant graph, such as the social network, knowledge graph and the user-item bipartite graph for recommendation. Here, it is non-trivial to keep the memory footprint under budget, especially when node and edge features have a large dimension.

The second dimension is whether they are static. Dynamic graphs are particularly important in generative models, for example to infer latent graph structure which is crucial in relational reasoning AI (Yang et al., 2018). Graph mutation occurs from time to time, e.g. by adding/removing nodes/edges (Jin et al., 2018; Li et al., 2018) or pooling (Ying et al., 2018b). Finally, no graphs are born giant, they grow to be (e.g. a knowledge graph); it is important to keep this *evolution* viewpoint in mind.

Obviously, the above categorization is not mutually exclusive and a complete model can embrace multiple aspects, depending on model dynamics, dataset type or optimization algorithm design.

3 Deep Graph Library

In this section, we focus on the core design of DGL's APIs and its system performance optimization.

		GNet	NGra	Euler	PyG	DGL
Message Passing	arbitrary ϕ^e	✓	✓	✓	✓	✓
	arbitrary ϕ^v	✓	✓	✓	✓	✓
	arbitrary $\bigoplus$	✓	✗	✓	✗	✓
Propagation Order	full	✓	✓	✗	✓	✓
	partial	✗	✗	✗	✓	✓
	random walk	✗	✗	✗	✗	✓
	sampling	✗	✗	✓	✗	✓
Graph Type	many & small	✓	✗	✗	✓	✓
	single & giant	✗	✓	✓	✗	✓
	dynamic	✗	✗	✗	✗	✓
System	multi-platform	✗	✗	✗	✗	✓

Table 1: DGL vs. GraphNet (GNet), NGra, Euler and Pytorch Geometric (PyG)

3.1 KEY USER-FACING APIS

DGL's central abstraction for graph data is `DGLGraph`. It is inspired by NetworkX (Hagberg et al., 2008) – a popular package for graph analytic, to which we maintain maximal similarity. `DGLGraph` stores node/edge features so that it can access or modify them at any time, even when the graph is mutated. Users query or modify these features with a dictionary-style interface, for example `g.ndata['x']` returns the x features of all nodes that are packed in one tensor.

DGL provides the following two basic primitives to perform computation on graphs:

$$\texttt{send}(\mathcal{E}, \phi^e), \quad \texttt{recv}(\mathcal{V}, \bigoplus, \phi^v) \tag{2}$$

Here, `send` and `recv` are message passing triggers on edge and node respectively. In addition, DGL extends them with the following:

User-defined function (UDF). DGL makes *no* assumption on what ϕ^e, ϕ^v and $\bigoplus$ in Equation 1 should be. They could be parameterized by neural network modules (e.g. `torch.nn.module`). Supporting arbitrary $\bigoplus$ efficiently is non-trivial due to varying node degrees. In DGL, we analyze the graph structure and process nodes with the same neighborhood size in a batch. We also perform aggressive optimization for the most popular UDF combinations, as we will explain later.

Active set. $\mathcal{E}$ and $\mathcal{V}$ define the set of edges and nodes that are triggered for message passing. Collectively, we call them **active set**. Computation can be repeatedly triggered on different active sets to propagate messages and update features along a walk on the graph or a preset path. DGL supports a variety of propagation orders using the active set design:

- subset node/edge propagation under certain traversal order (Tai et al., 2015; Jin et al., 2018);
- random walk (Ying et al., 2018a; Hamilton et al., 2017);
- active sets that are generated dynamically, including exploration algorithms on graph navigation (Nogueira & Cho, 2016; Das et al., 2017) and graph generative models (Li et al., 2018).

DGL also provides convenient utilities for message passing following common graph traversal orders (e.g. topological traversal, DFS, BFS, etc.).

Batching and Sampling APIs. In DGL, we expose batching as an API to collect a set of small graphs of different sizes into batches for efficient optimization. When faced with much larger graphs, many (Hamilton et al., 2017; Chen et al., 2017) propose to sample the graph to avoid full-graph propagation. In DGL, we provide graph sampling API in the form of data loader rendering sampled subgraphs. Optionally, these subgraphs can then be batched for efficient processing.

3.2 IMPLEMENTATION AND OPTIMIZATION

Graph storage management. Prior libraries including GraphNet and PyG require users to maintain graphs as sparse matrices and features separately as dense tensors. Although this is convenient for quick prototyping, many low-level system design choices are exposed to users: which sparse

Dataset $\lvert V\rvert$	$\lvert E\rvert$	Model	Accuracy	Time PyG	Time DGL	Memory PyG	Memory DGL
Cora		GCN	81.31 ± 0.88	**0.478**	0.666	1.1	1.1
3K	11K	GAT	83.98 ± 0.52	1.608	**1.399**	1.2	**1.1**
CiteSeer		GCN	70.98 ± 0.68	**0.490**	0.674	1.1	1.1
3K	9K	GAT	69.96 ± 0.53	1.606	**1.399**	1.3	**1.2**
PubMed		GCN	79.00 ± 0.41	**0.491**	0.690	1.1	1.1
20K	889K	GAT	77.65 ± 0.32	1.946	**1.393**	1.6	**1.2**
Reddit		GCN	93.46 ± 0.06	*OOM*	**28.6**	*OOM*	**11.7**
232K	114M						
Reddit-S		GCN	N/A	29.12	**9.44**	15.7	**3.6**
232K	23M						

Table 2: Training time (in seconds) for 200 epochs and memory consumption (GB).

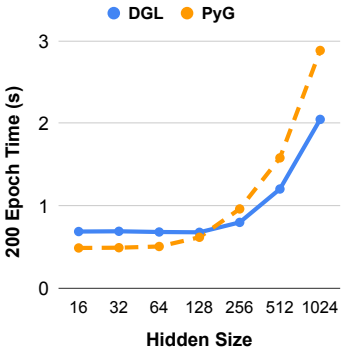

Figure 1: GCN training time on Pubmed with varying hidden size.

format (CSR or COO) to use, what is the order of stacking node/edge features, shall we store all the node/edge features in one tensor or many for heterogeneous graphs? By contrast, the DGL system (i.e, `DGLGraph`) manages the storage so that users are freed from the low-level decisions on how to manipulate graphs as tensors.

Tensorization and kernel fusion. Rarely does a user perform computation on a single node or edge. DGL stacks node/edge features so that the user-defined ϕ^e and ϕ^v can operate on a batch of nodes and edges. `send` and `recv` can also be fused together to avoid materializing messages. For example, when ϕ^e simply returns the sending node's feature and $\bigoplus$ is defined to be a sum, the overall computation is equivalent to a Sparse-Dense Matrix Multiply (SPMM) where the sparse matrix is the adjacency matrix of the subgraph defined by the edge set $\mathcal{E}$. In DGL, system automatically detects pattern of fusion and optimizes. It takes `send_and_recv`$(\mathcal{E}, \phi^e, \bigoplus, \phi^v)$ and gauges the possibility of fusion by assessing ϕ^e and $\bigoplus$. When fusion is possible it saves both time and memory, since messages typically do not need to be materialized.

4 Evaluation

Experimental setup. We focus on the semi-supervised setup for node classification (Kipf & Welling, 2017). Our datasets include three citation networks (Cora, Citeseer and Pubmed (Kipf & Welling, 2017)) as well as the larger scale Reddit posting graph (Hamilton et al. (2017)). For model selection, we follow the setup in the original papers and report the average performance from 10 rounds of experiments. All experiments are carried out on an AWS EC2 p3.2xlarge instance installed with one NVIDIA Tesla V100 GPU and 8 vCPUs.

Speed and memory efficiency. Table 2 evaluates the speed and memory consumption of training Graph Convolutional Network (Kipf & Welling, 2017) and Graph Attention Network (Veličković et al., 2018). We primarily compare DGL v0.3 against PyG v1.0.3 (Fey & Lenssen, 2019)[2]. As we can see from Table 2, DGL is consistently faster than PyG in training GAT. For GCN, DGL lags behind because it has a slightly higher python overhead cost in the wrapper codes (e.g. storage APIs) in the original setting. However when we increase the hidden layer size, DGL starts to out-perform PyG, as shown in Figure 1. GAT causes OOM for both DGL and PyG on Reddit so is not included.

DGL's superior performance is largely owing to kernel fusion. It brings both speed and memory advantage, in particular on larger and denser graphs such as Reddit graph whose average degree is around 500. On this dataset, PyG runs out of memory while DGL can complete training by using less than 12 GB memory. For further comparison, we randomly prune out 80% of the edges in Reddit graph (dubbed as Reddit-S). Under this setup, we can measure the PyG memory consumption. In this case, DGL is still **3**$\times$ faster and saves **77%** memory.

Handling large graphs with sampling. We test our sampling APIs on large graphs. Currently, its MXNet version runs on both large shared-memory NUMA machine as well as on multi-GPUs,

[2]PyG compared with DGL v0.2 in their paper which does not include DGL's the kernel fusion feature.

scaling upwards to 500M nodes. For Reddit graph, we verify that using the algorithm proposed by Chen et al. (2017), training GCN is roughly **3X** faster than the full graph propagation while converging to similar accuracy.

5 OUTLOOK

We present Deep Graph Library, a graph-oriented library built for deep learning on Graphs. On the ML side, we will continue to push ease of use, diversity and scale. On the system end, as the models grow deeper and larger, or structurally more complex, how to speed up them using modern hardware is another challenge; kernel fusion technique in DGL is only a stepping stone.

6 ACKNOWLEDGEMENT

This work is in part supported by Science and Technology Commission of Shanghai Municipality under award 17JC1404101, National Science Foundation (CNS-1816717) and NVIDIA AI Lab (NVAIL) at NYU. We also thank Tianqi Chen, Sheng Zha and Damon Deng for the help on open sourcing the project.

REFERENCES

Alibaba. Euler. `https://github.com/alibaba/euler`, 2019.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018a.

Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised community detection with line graph neural networks. 2018b.

Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. Learning steady-states of iterative algorithms over graphs. In *International Conference on Machine Learning*, pp. 1114–1122, 2018.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*, 2017.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. *CoRR*, abs/1903.02428, 2019.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864. ACM, 2016.

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. Star-transformer. *arXiv preprint arXiv:1902.09113*, 2019.

Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems*, pp. 2030–2041, 2018.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. Towards efficient large-scale graph neural network computing. *arXiv preprint arXiv:1810.08403*, 2018.

Rodrigo Nogueira and Kyunghyun Cho. End-to-end goal-driven web navigation. In *Advances in Neural Information Processing Systems*, pp. 1903–1911, 2016.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017.

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*, pp. 593–607. Springer, 2018.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

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

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*, 2019a.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019b.

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

Zhilin Yang, Bhuwan Dhingra, Kaiming He, William W Cohen, Ruslan Salakhutdinov, Yann LeCun, et al. Glomo: Unsupervisedly learned relational graphs as transferable representations. *arXiv preprint arXiv:1806.05662*, 2018.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *KDD*, 2018a.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pp. 4805–4815, 2018b.

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

A MODELS RELEASED BY DGL

Model released in DGL (* is currently in pull request):

- Message passing neural networks
  - GCN (Kipf & Welling, 2017)
  - GAT (Veličković et al., 2018)
  - SGC (Wu et al., 2019a)
  - R-GCN (Schlichtkrull et al., 2018)
  - LGNN (Chen et al., 2018b)
  - SSE (Dai et al., 2018)
- Using graph traversal order
  - TreeLSTM (Tai et al., 2015)
- Random walk and sampling on large graphs
  - GraphSAGE (Hamilton et al., 2017)
  - PinSAGE* (Ying et al., 2018a)
  - Variance Reduction (Chen et al., 2017)
- Generative models
  - Deep Generative Model for Graphs (Li et al., 2018)
  - DiffPool* (Ying et al., 2018b)
  - Junction Tree VAE (Jin et al., 2018)
- Non-local neural networks
  - Capsule Network (Sabour et al., 2017)
  - Transformer (Vaswani et al., 2017)
  - Star-Transformer* (Guo et al., 2019)