FaSt Inductive Representation Learning on Graphs

Muhammed Çavuşoğlu
Department of Computer Engineering
Bilkent University
Ankara, Turkey
m.cavusoglu@bilkent.edu.tr

Kemal Büyükkaya
Department of Computer Engineering
Bilkent University
Ankara, Turkey
kemal.buyukkaya@bilkent.edu.tr

***Abstract*—We extend the supervised version of an inductive reprentation learning framework for node classification task. We describe LSTM and Max-pooling aggregators that we implemented. We propose FaSt aggregator that captures different aspects of neighborhood feature sets, and learns large neighborhood feature sets. We conduct experiments on three citation network benchmark datasets, and compare our results against the state-of-the-art approaches. We show that, our implementation gives prediction accuracies comparable to the state-of-the-art. We also show that, FaSt aggregator yields faster and steadier convergence when the same early stopping criterion is applied based on validation accuracies.**

I. Introduction

Graphs are a ubiquitous data structure, used extensively within computer science and related fields. Social networks, biological protein-protein interaction networks, citation networks and many more can be modeled as graphs, which represent interactions (*i.e.*, edges) between individual units (*i.e.*, nodes). As a result, graphs allow relational knowledge about interacting entities to be efficiently stored and accessed [1].

However, graphs are not only useful as relational knowledge databases; they also play an important role in machine learning. Machine learning applications attempt to make predictions, or discover new patterns, using graph-structured data as feature information. For example, one might wish to predict the role of a person in a collaboration network, recommend friends to a user in a social network [2], or classify the role of a protein in a biological interaction graph [3].

The central problem in machine learning on graphs is incorporating information about the structure of the graph into the machine learning model. There are recent approaches that seek to *learn* representations that encode structural information about the graph. The idea behind these *representation learning* approaches is to learn a mapping that embeds nodes, or entire (sub)graphs, as points in a low-dimensional vector space, $\mathbb{R}^d$. The goal is to optimize this mapping so that geometric relationships in this learned space preserve the structure of the original graph. After this optimization, the learned embeddings can be used as feature inputs for downstream machine learning tasks [4].

The significance of representation learning is how it treats this problem of capturing structural information about the graph. Representation learning treats this problem as a machine learning task itself, and uses a data-driven approach to learn embeddings that encode graph structure; rather than using hand-engineered statistics to extract structural information [4].

However, previous works have focused on embedding nodes from a single fixed graph [5]–[8]. The inductive capability is essential for production machine learning systems, which operate on evolving graphs and constantly encounter unseen nodes. An inductive approach also provides generalization across graphs with the same form of features. In order to generalize, an inductive framework must learn to identify structural properties of a node's neighborhood that reveal both the node's local role in the graph, as well as its global position [3].

A general framework for inductive node embedding, called GraphSAGE (SAmple and aggreGatE), is proposed in [3]. Instead of training a distinct embedding vector for each node, it trains a set of *aggregator functions* that learn to aggregate feature information from a node's local neighborhood (Figure 1). Each aggregator function aggregates information from a different number of hops, or search depth, away from a given node.

In this paper, we extend the supervised version of GraphSAGE framework with mean aggregator [3] for node classification task. We describe LSTM and Max-pooling aggregators that we implemented based on [3]. We propose a new aggregator, called FaSt aggregator, that provides faster and steadier convergence when the same early stopping criterion is applied based on validation accuracies. In the following sections, we provide related work; describe our methodology, experiments that we conducted, and our results.

II. Related Work

GraphSAGE framework is conceptually related to node embedding approaches, general supervised approaches to learning over graph structure, and graph convolutional networks.

A. Factorization-based Embedding Approaches

There are different node embedding approaches that learn structural knowledge using random walk statistics and matrix factorization-based learning objectives. These methods are also related to traditional approaches that operate on graphs, including spectral clustering [9], multi-dimensional scaling [10], and PageRank algorithm [11]. They directly train node

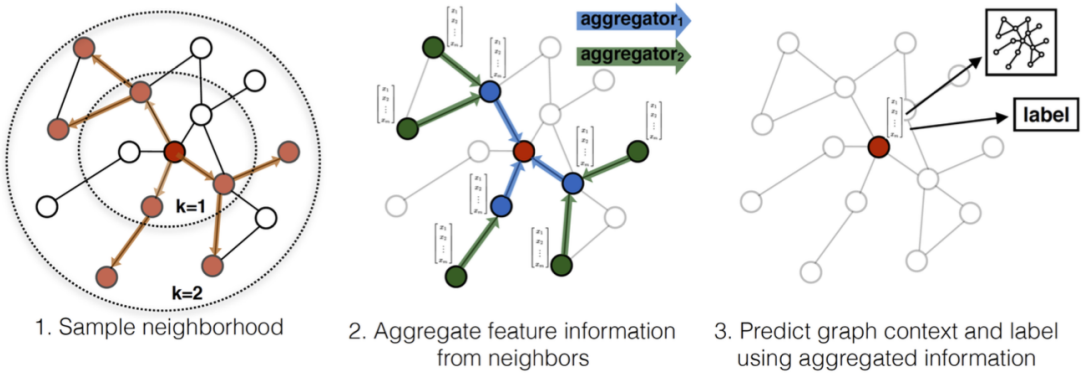

Fig. 1. Visual illustration of the GraphSAGE sample and aggregate approach [3]

embeddings for individual nodes. Therefore, they are inherently transductive, and require additional training to make predictions on new nodes. In other words, they can only generate embeddings for nodes that were present during the training phase [6].

B. Supervised Learning Over Graph Structure

Beyond node embedding approaches, there are a wide range of kernel-based, and neural network based approaches to supervised learning over graph structures [12]–[16]. The key distinction between these approaches and GraphSAGE framework is that they attempt to classify entire graphs (or subgraphs), while GraphSAGE aims to generate useful representations for individual nodes.

C. Graph Convolutional Networks

There are also convolutional neural network architectures that focus on learning the structural information of graphs [17]–[21]. The major downside of these methods is that they do not scale well to large graphs or are designed for whole-graph classification. The approach used in GraphSAGE framework is closely related to the graph convolutional network (GCN) presented in [21].

III. Methodology

The main idea behind GraphSAGE framework is that it learns how to aggregate feature information from a node's local neighborhood. We first describe the GraphSAGE embedding generation algorithm, which generates embeddings for nodes assuming that the GraphSAGE model parameters are already learned [3]. We then describe different aggregator functions.

A. Embedding Generation

In this section, we describe the embedding generation algorithm (Algorithm 1) proposed in [3]. In Algorithm 1, we assume that we have learned the parameters of K aggregator functions which aggregate information from node neighbors, and a set of weight matrices which are used to propagate information between different search depths.

The intuition behind Algorithm 1 is that at each iteration, nodes aggregate information from their local neighbors, and as this process iterates, nodes incrementally gain more information from further reaches of the graph [3]. Embedding generation process is described in Algorithm 1 for the case where the entire graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and features for all nodes $\mathbf{x}_v, \forall v \in \mathcal{V}$, are provided as input. In our implementation, this is generalized to the minibatch setting.

In Algorithm 1, we build up the representation for a node, represented by $\mathbf{h}^k$ for kth step. First, the node embeddings are initialized to be equal to the input node attributes. Then, at each iteration of the encoder algorithm, nodes aggregate the embeddings of their neighbors using an aggregation function (denoted by AGGREGATE in Algorithm 1) that operates over sets of vectors. After this aggregation, every node is assigned a new embedding that is equal to its aggregated neighborhood vector combined with its previous embedding from the last iteration. Finally, this combined embedding is fed through a fully connected layer and the process repeats. As the process iterates, the node embeddings contain information aggregated from further and further reaches of the graph. However, the dimensionality of the embedding remains constrained as the process iterates, so the encoder is forced to compress all neighborhood information into a low dimensional vector. After K iterations, the process terminates, and the final embedding vectors are output as the node representations [4]. Using this approach, we can generate embeddings for nodes that were not observed during the training phase, since we use the same aggregation function and weight matrices to generate embeddings for all nodes.

In the next section, we describe different aggregator architectures that can be used in Algorithm 1 (denoted by AGGREGATE in line 4).

B. Aggregator Architectures

Unlike machine learning over sentences or images, a node's neighbors have no natural ordering. Therefore, aggregator functions must operate over an unordered set of vectors. They must be invariant to the order of its arguments. In other

Algorithm 1: GraphSAGE embedding generation algorithm from [3]

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$;
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / ||\mathbf{h}_v^k||_2, \forall v \in \mathcal{V}$;
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

words, an aggregator function must be symmetric, so that the proposed neural network model can be trained and applied to arbitrarily ordered node neighborhood feature sets. We examined the aggregator functions below.

1) Mean Aggregator: The first aggregator function that we examined is the mean operator. Fundamentally, it takes the element-wise mean of node neighborhood feature sets [3]. Mean aggregator is conceptually similar to the convolutional propagation rule used in GCN framework [21].

2) LSTM Aggregator: We also examined an aggregator based on the LSTM architecture. Compared to the mean aggregator, LSTM-based aggregator provides larger expressive capability. On the other hand, since LSTMs process their input in a sequential manner, they are not permutation invariant; so that they are inherently not symmetric. In order to adapt LSTMs to operate on an unordered neighborhood feature set, we applied LSTMs to a random permutation of the node's neighbors.

3) Max-pooling Aggregator: We examined another aggregator which is based on the pooling approach. We initially feed each of the neighborhood feature sets independently through a fully-connected neural network. We then apply element-wise max-pooling operation to aggregate information across the neighborhood feature set. The motivation behind applying the max-pooling operator to each of the computed features is that, the model learns different aspects of the neighborhood feature set.

4) FaSt Aggregator: We propose a new aggregator that captures different aspects of the neighborhood feature sets (similar to max-pooling aggregator), and has the ability to learn large neighborhood feature sets (similar to LSTM aggregator). FaSt aggregator is divided into two branches: a fully connected network branch and an LSTM branch. In fully connected network branch, we feed each of the neighborhood feature sets independently through a fully connected network with three hidden layers. We apply ReLU activations to the outputs of the hidden units. We also apply Layer Normalization [22] between the hidden layers. In LSTM branch, we apply LSTMs to the same neighborhood feature sets. In order to combine the outputs of these two branches, we apply element-wise max-pooling operation to the resulting vectors of each branch independently, and take the mean of the results coming from the max-pooling operation.

IV. Experiments

In this section, we describe the three citation network dataset that we used, our experimental setup, and the three baselines that we compared our results against.

A. Dataset

We conduct our experiments on three citation network benchmark datasets-Cora, Citeseet and PubMed [23]. Datasets are summarized in Table I, and detailed descriptions are given below.

1) Cora: Cora dataset contains 5,429 citation links (edges) and 2,708 documents (nodes) classified into 7 classes. Citation links are treated as undirected edges and a binary, symmetric adjacency matrix is constructed. Each document has a class label and a 1,433 dimensional sparse bag-of-words feature vector.

2) PubMed: PubMed dataset contains 44,338 links and 19,717 documents classified into 3 classes. Each document has a class label and a 500 dimensional sparse bag-of-words feature vector.

3) Citeseer: Citeseer dataset contains 4,732 links and 3,327 documents classified into 6 classes. Each document has a class label and a 3,703 dimensional sparse bag-of-words feature vector.

B. Experimental Setup

We classify research topics using the Cora and Citeseer datasets, and categorize academic papers with the PubMed dataset.

We use our re-implemented and extended version of the reference PyTorch implementation[1] of GraphSAGE with mean aggregator. We also test Max-pooling and LSTM aggregators

[1]https://github.com/williamleif/graphsage-simple

TABLE I
OVERVIEW OF THE THREE DATASETS

Dataset	Nodes	Edges	Classes	Features	Training/Validation/Test (as in [24])
Cora	2708	5429	7	1433	1208/500/1000
PubMed	19717	44338	3	500	18217/500/1000
Citeseer	3327	4732	6	3703	1827/500/1000

that we implemented as described in [3], and our proposed FaSt aggregator.

We use Adam optimizer [25] with learning rate of 0.001 for Cora and Citeseer experiments, and 0.01 for PubMed experiments. We apply early stopping based on validation accuracy scores and train 2000 epochs at most.

C. Baselines

We compare our results against three state-of-the-art baselines: GCN[2] [21], FastGCN [24], and GraphSAGE with mean aggregator implementation by [3]. We use the results reported in [26] (November 2018 paper).

V. RESULTS AND DISCUSSION

Test prediction accuracies of the three state of the art methods, and our GraphSAGE implementation with four different aggregators on the three citation networks are given in Table II. For each dataset (each column of Table II), the best accuracy result of the state of the art methods, and the best accuracy result of our methods are indicated in bold.

TABLE II
TEST PREDICTION ACCURACIES

Methods	Cora	PubMed	Citeseer
GCN	**86.3**%	86.8%	**77.8**%
FastGCN	85.0%	**88.0**%	77.6%
GraphSAGE-mean	82.2%	87.1%	71.4%
Extended GraphSAGE-FaSt	**84.1**%	83.8%	71.2%
Extended GraphSAGE-mean	82.1%	**86.0**%	**72.8**%
Extended GraphSAGE-LSTM	83.0%	83.6%	69.8%
Extended GraphSAGE-maxpool	82.8%	81.6%	70.7%

Based on the results in Table II, we can say that prediction accuracies of our methods are comparable to the state-of-the-art. There is a difference of around 2% for Cora and Citeseer datasets, and of 5% for PubMed dataset. Difference in accuracy could be due to other optimization tricks that were not mentioned in respective papers.

Comparison of Different Aggregator Architectures

We observe that none of the four aggregator functions implemented give significantly better prediction accuracy when compared to the other three. This can be validated using Table II results; as for Cora, FaSt aggregator gives the best result, and for PubMed and Citeseer, the mean aggregator gives the best result among our methods.

However, FaSt aggregator provides faster and steadier convergence. We can observe this based on the validation accuracy and training loss plots given in Figure 2 for Cora dataset.

[2]https://github.com/tkipf/gcn

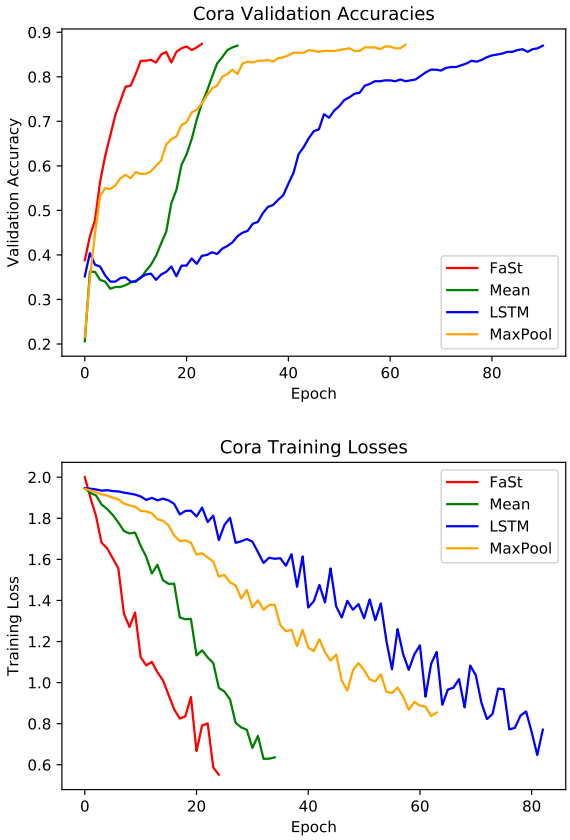

Fig. 2. Validation accuracies and training loses per epoch for Cora dataset

When the same early stopping criterion based on validation accuracy is applied to all four aggregators running on Cora, FaSt aggregator converges the fastest and early stops at around $20th$ epoch. It also yields the lowest training loss before other aggregators.

Similar case can be observed in the validation accuracy and training loss plots given in Figure 3 for PubMed dataset.

When the same early stopping criterion based on validation accuracy is applied to all four aggregators running on PubMed, FaSt aggregator converges the fastest and early stops at around $12th$ epoch.

Similar case can also be observed in the validation accuracy and training loss plots given in Figure 4 for Citeseer dataset.

When the same early stopping criterion based on validation accuracy is applied to all four aggregators running on Citeseer, FaSt aggregator converges the fastest and early stops at around $15th$ epoch.

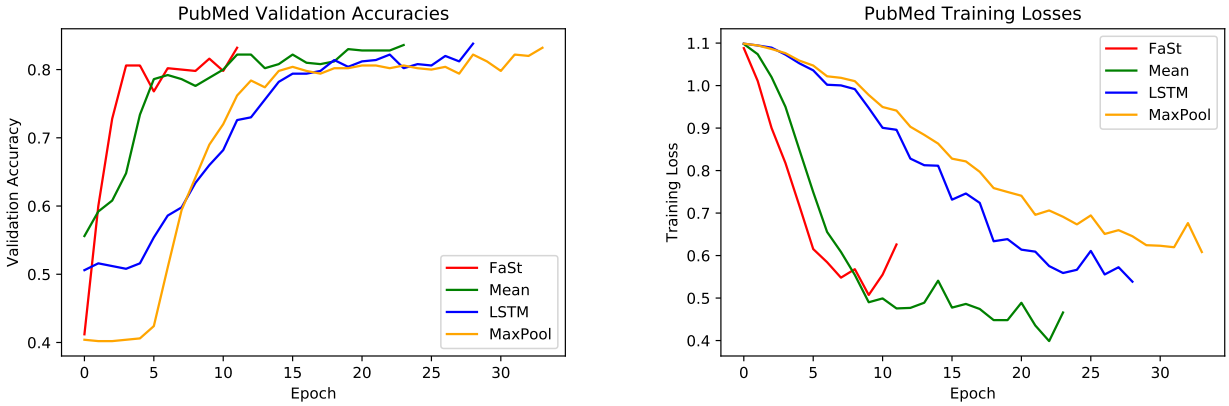

Fig. 3. Validation accuracies and training loses per epoch for PubMed dataset

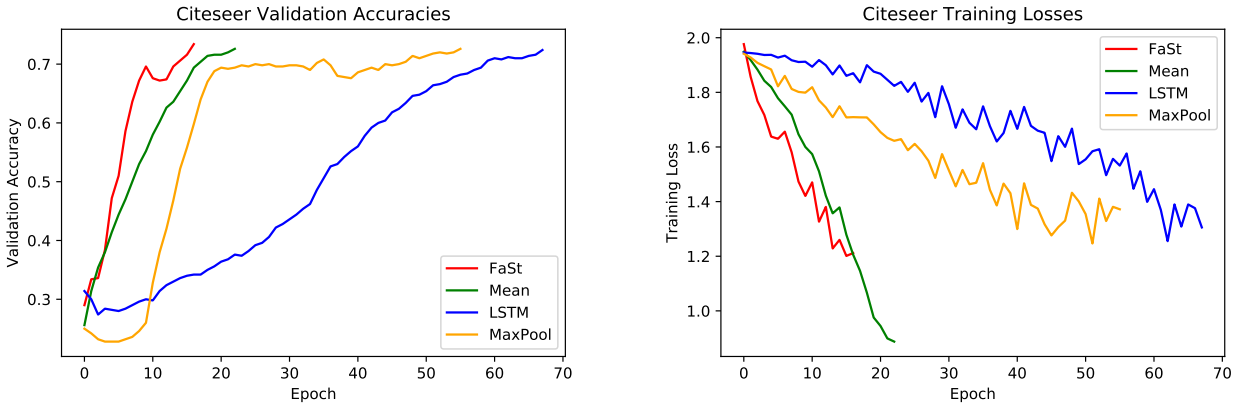

Fig. 4. Validation accuracies and training loses per epoch for Citeseer dataset

VI. Conclusion

We extended the supervised version of GraphSAGE framework with mean aggregator [3] for node classification task. We implemented LSTM and Max-pooling aggregators as described in [3]. We proposed a new aggregator, called FaSt aggregator, that captures different aspects of the neighborhood feature sets, and learns large neighborhood feature sets. We conducted experiments on three citation network benchmark datasets, namely Cora, Citeseet and PubMed [23]; and compared our results against recent state-of-the-art methods. We showed that, the extended version of GraphSAGE that we implemented gave prediction accuracies comparable to the results of the recent state-of-the-art methods (Table II). We also showed that, FaSt aggregator yields faster and steadier convergence when the same early stopping criterion is applied based on validation accuracies.

References

[1] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, pp. 1:1–1:39, Feb. 2008.

[2] L. Backstrom and J. Leskovec, "Supervised random walks: Predicting and recommending links in social networks," in *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, (New York, NY, USA), pp. 635–644, ACM, 2011.

[3] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[4] W. L. Hamilton, Z. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, pp. 52–74, 2017.

[5] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, (New York, NY, USA), pp. 891–900, ACM, 2015.

[6] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 855–864, ACM, 2016.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, (New York, NY, USA), pp. 701–710, ACM, 2014.

[8] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, (Republic and Canton of Geneva, Switzerland), pp. 1067–1077, International World Wide Web Conferences Steering Committee, 2015.

[9] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, (Cambridge, MA, USA), pp. 849–856, MIT Press, 2001.

[10] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, pp. 1–27, Mar 1964.

[11] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation

ranking: Bringing order to the web.," Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[12] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 2702–2711, JMLR.org, 2016.

[13] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 729–734 vol. 2, July 2005.

[14] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," in *Proceedings of ICLR'16*, April 2016.

[15] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *Trans. Neur. Netw.*, vol. 20, pp. 61–80, Jan. 2009.

[16] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, Nov. 2011.

[17] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[18] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2224–2232, Curran Associates, Inc., 2015.

[19] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 3844–3852, Curran Associates, Inc., 2016.

[20] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 2014–2023, JMLR.org, 2016.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[22] J. Ba, R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[23] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[24] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," *CoRR*, vol. abs/1801.10247, 2018.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[26] L. Zhang, H. Song, and H. Lu, "Graph node-feature convolution for representation learning," *CoRR*, vol. abs/1812.00086, 2018.