AUTOMATED GRAPH LEARNING VIA POPULATION BASED SELF-TUNING GCN

Ronghang Zhu, *Zhiqiang Tao†, Yaliang Li‡, Sheng Li**(✉)

*Department of Computer Science, University of Georgia, Athens, GA
†Department of Computer Science and Engineering, Santa Clara University, Santa Clara, CA
‡Alibaba Group, Bellevue, WA
{ronghangzhu, sheng.li}@uga.edu, ztao@scu.edu, yaliang.li@alibaba-inc.com

ABSTRACT

Owing to the remarkable capability of extracting effective graph embeddings, graph convolutional network (GCN) and its variants have been successfully applied to a broad range of tasks, such as node classification, link prediction, and graph classification. Traditional GCN models suffer from the issues of overfitting and oversmoothing, while some recent techniques like DropEdge could alleviate these issues and thus enable the development of deep GCN. However, training GCN models is non-trivial, as it is sensitive to the choice of hyperparameters such as dropout rate and learning weight decay, especially for deep GCN models. In this paper, we aim to automate the training of GCN models through hyperparameter optimization. To be specific, we propose a self-tuning GCN approach with an alternate training algorithm, and further extend our approach by incorporating the population based training scheme. Experimental results on three benchmark datasets demonstrate the effectiveness of our approaches on optimizing multi-layer GCN, compared with several representative baselines.

Index Terms— Graph Neural Networks, Graph Learning, Hyperparameter Optimization.

1. INTRODUCTION

Graph-structured data are ubiquitous in various real-world applications, which promotes the demand of expanding deep learning techniques to graphs [1]. Many approaches have been proposed to learn node feature representations by investigating graph convolutional networks (GCN) [2, 3, 4, 5]. However, these GCN models are usually no deeper than three or four layers. Recently, several attempts have been proposed to explore deeper GCN models, such as incorporating residual/dense connections and dilated convolutions to build deeper GCN models [6, 7], and adopting the idea of DropEdge [8] to solve overfitting and oversmoothing problems in deeper GCN. Most of these approaches focus on designing an appropriate deeper GCN structure yet ignoring the importance of hyperparameter choices to train deeper GCN models.

Recently, automatically optimizing hyperparameters has yielded remarkable results on many machine learning tasks. There are mainly two categories: One is hyperparameter configuration search, like random search [9], grid search [10] and Hyperband [11], which optimizes hyperparameters in fixed values during training process. The other is hyperparameter schedule search such as self-tuning networks (STN) [12] and population based training (PBT) [13], which enable hyperparameters to change in each training iteration. To the best of our knowledge, the hyperparameter optimization problem for graph neural networks has not been studied yet.

In this paper, we propose a novel automated graph learning algorithm to investigate deeper GCN models. Different from existing works on graph neural architecture search [14, 15, 16], our work focuses on automatically tuning hyperparameters with given GCN architectures. The major contributions of this paper are summarized as follows.

- We propose to solve the automated graph learning problem from a new perspective, *i.e.*, through hyperparameter optimization, which provides a complementary direction to graph neural architecture search.

- We design and develop self-tuning GCN (ST-GCN) by incorporating hypernets [12] in each graph convolutional layer, enabling a joint optimization over GCN model parameters and its hyperparameters. The proposed approach can be flexibly extended to many existing GCN models [17, 3, 4].

- We further adopt a population based training framework to self-tuning GCN, which alleviates local minima problem by exploring hyperparameter space globally.

- We conduct extensive experiments to demonstrate the effectiveness of the proposed approaches on three benchmark datasets.

2. RELATED WORK

Graph Neural Networks (GNNs) have been a mainstream technique to squeeze complex graph-structured data into com-

pact and low-dimensional embedding representations [1, 17, 18, 19, 20]. Roughly, it could be separated into two categories: spectral-based GNNs [21, 2] and spatial-based GNNs [3, 4]. The former develops graph convolution operations in the vein of spectral graph theory; the latter instantiates convolution on spatial domain, relying more on neighborhood sampling, message passing, and feature aggregation. To name a few, graph convolutional networks (GCN) [2] and graph attention networks (GAT) [3] are two representative spectral and spatial-based methods, respectively, which are widely used as building modules in other graph learning frameworks. In this paper, we focus our study on the first category, with a particular interest in the impact of hyperparameters on GCN.

Hyperparameter Optimization (HPO) [22] lies in the core task of AutoML, which aims to optimize the model performance by automatically searching feasible hyperparameter configurations or schedules. Some representative HPO methods, yet not limited to, include grid/random search [9], Hyperband and successive halving [11], hypergradient based method [23], and Bayesian optimization [24]. Recent research [8] has shown that different hyperparameters, such as learning rates, dropout, weight decay, etc., largely impact the model performance of various GNN architectures. Thus, it is a promising direction to incorporate HPO into GNNs to enable automated learning. In light of this, we develop a Self-tuning GCN (ST-GCN) model with population-based training, inspired by the recent hyperparameter scheduling methods [13, 12, 25]. To our best knowledge, this is the first attempt to propose a GNN-specific hyperparameter optimization algorithm.

Automated Graph Learning has emerged as an important research problem that combines AutoML and GNNs. Similar to neural architecture search (NAS), existing automated graph learning methods mainly target to explore an optimal GNN architecture from a pre-defined network configuration space [14, 15]. The searching space is generally defined by GNN structures, including network depth, aggregation and activation functions, etc., and the search processing is governed by a controller model to optimize the performance on the validation set. Following this line, a series of graph neural architecture search (GNAS) methods have been proposed recently, implemented by reinforcement learning [14, 15, 16] and evolution algorithm [26, 27]. Unlike GNAS, the proposed ST-GCN studies "automation" from a new perspective, *i.e.*, hyperparameter optimization, which automatically tunes hyperparameters for pre-defined network architecture, and thus serves as a complementary direction towards automated graph learning.

3. METHODOLOGY

3.1. Preliminary

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote an undirected graph with N nodes $v_i \in \mathcal{V}$ $(i = 1, \cdots, N)$ and a number of edges $(v_i, v_j) \in \mathcal{E}$. The adjacency matrix of graph $\mathcal{G}$ is denoted by $A \in \mathbb{R}^{N \times N}$, where $A_{ij} = 1$ if there is an edge between v_i and v_j. We consider the node classification task [28] on graph $\mathcal{G}$, and use $\mathcal{Y}$ with $y_i \in \mathbb{R}$ $(i = 1, \cdots, N)$ to denote the node labels. Moreover, we define a graph learning model (e.g., GCN) as $f(\cdot; \theta, \lambda) : \mathcal{V} \to \mathcal{Y}$, which is parameterized by the model parameters $\theta \in \mathbb{R}^p$ (e.g., weights ans biases) and hyperparameters $\lambda \in \mathbb{R}^q$ (e.g., dropout rate and weight decay).

For node classification, the graph learning model $f(\cdot; \theta, \lambda)$ can be optimized by solving:

$$\mathcal{L}(\theta, \lambda) = \mathbb{E}_{(v,y) \in \mathcal{D}} \big[\ell(f(v; \theta, \lambda), y) \big], \tag{1}$$

where $\ell(\cdot, \cdot)$ refers to a loss function and $\mathcal{D}$ represents a training set $\mathcal{D}_{trn}$ or a validation set $\mathcal{D}_{val}$. Upon Eq. equation 1, we can see the loss value depends on both the model parameters θ and the hyperparameters λ. Traditionally, the selection of hyperparameters λ is an iterative manner with trial and error required profound knowledge of machine learning algorithms and statistics. In this paper, we aim to design an approach to automatically choose optimal hyperparameters for graph learning models.

The model $f(v; \theta, \lambda)$ in Eq. equation 1 could be implemented by various graph learning approaches, such as the traditional graph embedding methods and the recent graph neural networks. In this paper, we focus on graph convolutional networks (GCN) [2]. However, it is worth noting that, the proposed hyperparameter optimization method can be easily adapted to other graph neural networks. Given an input graph characterized by a normalized adjacency matrix $\hat{A}$ and a node feature matrix $H^{(0)} = X$, GCN updates the node embeddings by using the following layer-wise propagation rule:

$$H^{(l+1)} = \sigma\Big(\hat{A} H^{(l)} W^{(l)} \Big), \tag{2}$$

where $\hat{A} = \hat{D}^{-1/2}(A + I)\hat{D}^{-1/2}$ is the normalized adjacency matrix, I is an identity matrix, and $\hat{D}$ is the degree matrix of $A + I$. $W^{(l)} \in \mathbb{R}^{M_l \times M_{l+1}}$ is the learnable weight matrix in the l-th layer with M_l refers to the feature dimension of $H^{(l)}$. $H^{(l)} = \{h_1^{(l)}, h_2^{(l)}, \ldots, h_N^{(l)}\}$ is the hidden feature matrix with $h_i^{(l)}$ as the hidden feature for node v_i. $\sigma(\cdot)$ denotes an activation function such as the ReLu function.

Then, the graph learning model with two-layer GCN for node classification is defined as

$$f(\mathcal{V}; \theta, \lambda) = \text{softmax}\Big(\hat{A} \text{ReLu}\Big(\hat{A} \mathcal{V} W^{(0)} \Big) W^{(1)} \Big). \tag{3}$$

As mentioned in [2], overfitting is one of the main obstacles to build deep GCN model for node classification. To

alleviate this issue, DropEdge [8] is proposed to randomly drop out a certain part of edges in the graph, which is defined as

$$A_{drop} = A - A'. \tag{4}$$

Here A' denotes a random subset of edges from original A, A_{drop} refers to the resulting adjacency matrix after dropped edges. Replaced $\hat{A}$ with $\hat{A}_{drop}$ in deep GCN model for propagation can prevent overfitting problem, where $\hat{A}_{drop}$ is the re-normalized A_{drop}. In the following, we consider four-layer and eight-layer GCN models with DropEdge [8] for node classification on a graph.

3.2. Self-Tuning GCN

We propose a self-tuning GCN (ST-GCN) approach to guide the hyperparameter search of GCN models. In particular, we define $\hat{\theta}(\lambda) : \mathbb{R}^q \to \mathbb{R}^p$ as the response function of hyperparameter λ to approximate the GCN model parameter θ, i.e., $\hat{\theta}(\lambda)$ is a mapping from λ to optimal parameters. For a given GCN layer with the weight matrix $W \in \mathbb{R}^{M_{in} \times M_{out}}$ and bias $b \in \mathbb{R}^{M_{out}}$, we define the affine transformation of hyperparameters λ as

$$\hat{W}(\lambda) = W + W_\lambda \odot_{row} C_W(\lambda) \text{ and } \hat{b}(\lambda) = b + b_\lambda \odot C_b(\lambda), \tag{5}$$

where the dimensions of W_λ and b_λ are the same as W and b, respectively. $C_W(\lambda) \in \mathbb{R}^{M_{out}}$ and $C_b(\lambda) \in \mathbb{R}^{M_{out}}$ are scaled embeddings by linearly transforming λ, i.e., $C_w(\lambda) = e_w \lambda$ and $C_b(\lambda) = e_b \lambda$. $\odot_{row}$ denotes the row-wise rescaling and $\odot$ indicates element-wise multiplication. Thus the total parameters of the GCN model are $\hat{\theta}(\lambda) = \{\hat{W}(\lambda), \hat{b}(\lambda)\}$.

As $\hat{\theta}(\lambda)$ captures changes such as shifting and scaling in θ induced by λ, we reformulate Eq. equation 1 with $\hat{\theta}(\lambda)$ as

$$\mathcal{L}(\hat{\theta}(\lambda), \lambda) = \mathbb{E}_{\lambda \sim P(\lambda), (v,y) \in \mathcal{D}}\left[\ell(f(v; \hat{\theta}(\lambda), \lambda), y)\right]. \tag{6}$$

Here, $P(\lambda) = P(\lambda|\epsilon)$ represents a log-uniform distribution over hyperparameter λ. ϵ control the sccale of the hyperparameter distribution, which contains the bounds of the ranges of λ. $\hat{\theta}(\lambda)$ can capture the best-response over the samples and the shape locally around the hyperparameter values. We vary the distribution $P(\lambda|\epsilon)$ with training iterations. To prevent $P(\lambda|\epsilon)$ from collapsing to a degenerate distribution, we add an entropy regularization item $\mathbb{H}[\cdot]$ weighted by $\tau \in \mathbb{R}_+$. The objective function in Eq. equation 6 becomes:

$$\mathcal{L}(\hat{\theta}(\lambda), \lambda) = \mathbb{E}_{\lambda \sim P(\lambda|\epsilon), (v,y) \in \mathcal{D}}\left[\ell(f(v; \hat{\theta}(\lambda), \lambda), y)\right] - \tau \mathbb{H}[P(\lambda|\epsilon)]. \tag{7}$$

3.3. Alternate Training Algorithm

To optimize the objective functions of ST-GCN in Eq. equation 6 and Eq. equation 7, we follow the alternate training procedure proposed by [12], which includes two steps,

---

Algorithm 1 Self-Tuning GCN

Require: Hyperparameter λ, GCN model parameter θ, distribution scale ϵ response function $\hat{\theta}(\cdot)$, learning rate η_λ, η_θ and η_ϵ.
Ensure: GCN model parameter θ, hyperparameter λ
1: **while** not converged **do**
2:     **for** $i = 1, \ldots, T_{trn}$ **do**
3:         $\theta \leftarrow \theta - \eta_\theta \frac{\partial \mathcal{L}_{trn}}{\partial \theta}$
4:     **end for**
5:     **for** $i = 1, \ldots, T_{val}$ **do**
6:         $\lambda \leftarrow \lambda - \eta_\lambda \frac{\partial \mathcal{L}_{val}}{\partial \lambda}, \epsilon \leftarrow \epsilon - \eta_\epsilon \frac{\partial \mathcal{L}_{val}}{\partial \epsilon}, \epsilon \in P(\lambda|\epsilon)$
7:     **end for**
8: **end while**
9: **return** θ, λ

---

i.e., $ModelTraining$ and $HyperTraining$. In particular, we use $\mathcal{L}_{trn}$ to denote the $ModelTraining$ loss on training set $\mathcal{D}_{trn}$ following Eq. equation 6, and use $\mathcal{L}_{val}$ to denote the $HyperTraining$ loss on validation set $\mathcal{D}_{val}$ following Eq. equation 7.

In the $ModelTraining$ step, we adopt a stochastic gradient update of θ to minimize Eq. equation 6 with sampling $\lambda \sim P(\lambda|\epsilon)$. Specifically, θ is updated by

$$\theta_{(t)} \leftarrow \hat{\theta}_{(t-1)}(\lambda_{(t-1)}) - \eta_\theta \nabla \theta, \tag{8}$$

where η_θ is the learning rate, $\nabla \theta = \frac{\partial \mathcal{L}_{trn}}{\partial \theta}$ is the gradient of model parameter.

In the $HyperTraining$ step, we make a stochastic gradient update of λ and ϵ to minimize Eq. equation 7. In detail, λ is updated by

$$\lambda_{(t)} \leftarrow \hat{\lambda}_{(t-1)} - \eta_\lambda \nabla \lambda, \tag{9}$$

where η_λ is the learning rate, and $\nabla \lambda$ is the hypergradient given by

$$\nabla \lambda = \frac{\partial \mathcal{L}_{val}(\hat{\theta}(\lambda), \lambda)}{\partial \theta} \frac{\partial \theta}{\partial \lambda} + \frac{\partial \mathcal{L}_{val}(\hat{\theta}(\lambda), \lambda)}{\partial \lambda}. \tag{10}$$

The computation in Eq. equation 10 is mainly determined by response function $\hat{\theta}(\lambda)$, which is memory-efficient and flexible to compute [12]. We summarize the procedures of ST-GCN in Algorithm 1.

3.4. Population based Self-Tuning GCN

Our ST-GCN approach mainly focuses on computing hypergradients by designing a differentiable response function for hyperparameter, which is a non-convex optimization problem and may get into local minima [25]. To address this potential issue with ST-GCN, we propose a population based self-tuning GCN (PST-GCN) approach, inspired by the recent success of population based training [13, 25]. The population based training (PBT) could supply abundant multiplicity to globally seek hyperparameters throughout the hyperparameter space, which

3

Algorithm 2 Population based Self-Tuning GCN

Require: An initialized population of GCN models $\mathcal{S} = \left\{ s^k \right\}_{k=1}^{K}$, where $s^k = f(\cdot; \theta^k, \lambda^k)$.

Ensure: GCN model parameter θ, hyperparameter λ

1: **for** $(\theta, \lambda, s, t) \in \mathcal{S}$ (asynchronously in parallel) **do**
2:    **while** not end of training step **do**
3:      $\theta \leftarrow ModelTraining(\mathcal{D}_{trn}; \theta, \lambda)$
4:      $(\lambda, \epsilon) \leftarrow HyperTraining(\mathcal{D}_{val}; \theta, \lambda)$
5:      $acc_s \leftarrow eval(\mathcal{D}_{val}, s)$
6:      **if** $ready(s, t, \mathcal{S})$ **then**
7:        $(\theta', \lambda') \leftarrow Exploitation(\theta, \lambda, s, \mathcal{S})$
8:        **if** $\theta \neq \theta'$ **then**
9:          $(\theta, \lambda) \leftarrow Exploration(\theta', \lambda')$, $acc_s \leftarrow eval(\mathcal{D}_{val}, s)$
10:        **end if**
11:      **end if**
12:      update $\mathcal{S}$ with new $(\theta, \lambda, s, t+1)$
13:    **end while**
14: **end for**
15: **return** θ, λ with the highest acc_s of s in $\mathcal{S}$

---

employs a population of agent models to search different hyperparameter combinations and update hyperparameters with a mutation operation. PBT is a good complementary to ST-GCN, as it could help overcome the local minima issue.

Given a population of agent models $\mathcal{S}_{(t)} = \left\{ s^k_{(t)} \right\}_{k=1}^{K}$ at the t-th training step, where $s^k_{(t)}$ denotes the k-th agent model w.r.t $f(\cdot; \theta, \lambda)$, and K refers to the population size. The searching process of Population based Self-Tuning GCN (PST-GCN) includes three types of operations as follows:

- **Training step** evaluates the accuracy on the validation dataset, $eval(\mathcal{D}_{val}, s^k_{(t)})$, and updates $\theta^k_{(t-1)}$ to $\theta^k_{(t)}$, $\lambda^k_{(t-1)}$ to $\lambda^k_{(t)}$ and $\epsilon^k_{(t-1)}$ to $\epsilon^k_{(t)}$. The training step has a fixed number of epochs. After the training step, the agent $s^k_{(t)}$ is ready for exploitation and exploration.

- **Exploitation** operation exploits the population of agent models $\mathcal{S}_{(t)}$ by dividing it into three subsets of top, $middle$ and $bottom$ in terms of the accuracy of validation. The parameters and hyperparameters in the $bottom$ agent models are replaced by the top ones.

- **Exploration** operation maintains the top and $middle$ agents unchanged, and randomly perturb the hyperparameters of $bottom$ agents.

With the above three operations, the proposed PST-GCN can have a balance between local and global search to overcome the potential local minima issue in ST-GCN. Algorithm 2 summarizes the main procedures of our PST-GCN approach.

4. EXPERIMENTS

Datasets. We employ three benchmark citation network datasets for experiments, including Cora, Citeseer and Pubmed [29]. Each document contains a sparse bag-of-words feature vector, and there is a list of citation links between documents. We treat the documents as nodes and citation links as undirected edges. The statistics of these three datasets are summarized in Table 1. Label rate denotes the percentage of labeled nodes among all nodes for model training. We follow the settings in [2] and use the full-supervised training fashion on all datasets in experiments.

Table 1: Statistics of three benchmark graph datasets.

Dataset	Nodes	Edges	Classes	Features	Label Rate
Cora	2,708	5,429	7	1,433	0.052
Citeseer	3,327	4,732	6	3,703	0.036
Pubmed	19,717	44,338	3	500	0.003

Experiment Setting. We adopt two different backbones of GCN with 4 layers and 8 layers separately. The hyperparameters considered in our experiments include variational dropout rates for hidden state, edge dropout rate [8] and weight decay. The numbers of hyperparameters are 5 and 9 for the 4-layer and 8-layer GCN model, respectively.

Baselines. We compare the proposed methods ST-GCN and PST-GCN with three representative baselines including: (1) random search (RS): the best single model after 200 trials; (2) Hyperband (HB) [11]: the best single model after 200 trials; and (3) PBT [13]: the best single model in 200 agent models.
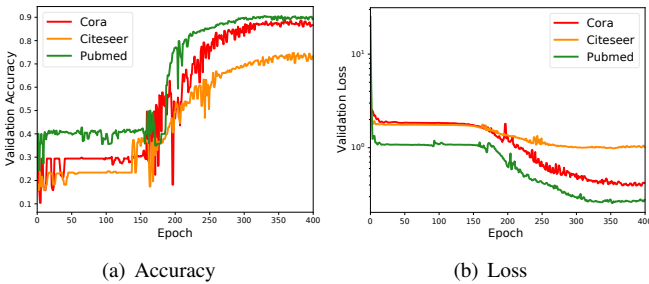
Implementation Details. We train the 4-layer and 8-layer GCN models with 128 hidden units per layer with all the above methods. Each model is trained for a maximum of 400 epochs. For PBT and PST-GCN, we set 200 epochs as warm up and 200 epochs as training iterations. After warm-up, we take one training epoch as one training step, and then perform exploitation and exploration after each training step. For ST-GCN and PST-GCN, we use an alternating training schedule to update the model parameters for two epochs on the $\mathcal{D}_{trn}$ and then update the hyperparameters for one epoch on the $\mathcal{D}_{val}$. We adopt the Adam optimizer for model training. For fixed hyperparameter baselines, i.e., RS and HB, the learning rate is set to 0.01 on the Cora and Pubmed datasets, and 0.09 on the Citeseer dataset, which follows the same settings in [8]. For the hyperparameter schedule baseline PBT and our approaches, we set the learning rate to 0.0005 on the Cora and Citeseer datasets, and 0.005 on the Pubmed dataset. The search spaces for the hyperparameters are as follows: dropout rates are in the range $[0, 0.9]$, edge dropout rates [8] are in the range $[0, 0.9]$, and weight decay is in the range $[10^{-6}, 10^{-2}]$.

Results and Analysis. Performance metric is the classification accuracy on testing set in percent, and results are summarized in Table 2. From it, we can observe that the proposed methods, ST-GCN and PST-GCN, obtain the best accuracy in

Table 2: Node classification accuracy (in percentage) of our approaches (ST-GCN and PST-GCN) and three baselines (RS, HB and PBT) on Core, Citeseer and Pubmed datasets.

Dataset	Layer	Method				
		RS	HB	PBT	ST-GCN	PST-GCN
Cora	4	85.3	84.5	82.9	86.2	**86.9**
	8	83.5	82.9	84.4	85.6	**87.0**
Citeseer	4	76.3	76.5	74.3	**79.3**	79.0
	8	74.9	74.8	73.8	78.1	**78.6**
Pubmed	4	**90.0**	89.3	88.4	89.6	89.8
	8	87.4	86.5	88.2	89.4	**90.4**

5 out of 6 experiments. Furthermore, compared with baselines on 4-layer and 8-layer GCN models, ST-GCN and PST-GCN have better advantages on 8-layer GCN models, which is due to the mighty power of gradient-based HPO in sophisticated hyperparameter space. Note that comparing with ST-GCN, PST-GCN can boost performance significantly on 8-layer GCN models, showing that population-based training can help alleviate the local minima problem in ST-GCN. Further, Fig. 1(a) and Fig. 1(b) show the validation accuracy and loss of ST-GCN over training epochs on 8-layer GCN model. We can observe that the validation accuracy is continuously improved along with the training epoch, while the validation loss curves are continuously decreased, even after a large number of epochs, which validates the effect of ST-GCN on alleviating overfitting in deeper GCN models.

(a) Accuracy　　　　　(b) Loss

Fig. 1: Experiments on Core, Citeseer and Pubmed dataset with 8-layer GCN. (a) The validation accuracy of ST-GCN over training epochs. (b) The validation loss of ST-GCN over training epochs.

5. CONCLUSIONS

In this paper, we study the automation of graph learning from the perspective of hyperparameter optimization, which is complementary to the existing GNN architecture search. To fulfill this goal, we propose a self-tuning GCN by jointly optimizing GCN model parameters and the hyperparameters, which can alleviate the overfitting in deeper GCN models. We further incorporate the population-based training to alleviate the local minima problem in ST-GCN, which provides a global hyperparameter search. Experimental results on three benchmark datasets demonstrate the benefit of the proposed population based self-tuning GCN method.

6. REFERENCES

[1] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[2] Thomas N Kipf and Max Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[3] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio, "Graph attention networks," in *ICLR*, 2018.

[4] Will Hamilton, Zhitao Ying, and Jure Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.

[5] Xiaodong Jiang, Ronghang Zhu, Pengsheng Ji, and Sheng Li, "Co-embedding of nodes and edges with graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[6] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem, "Deepgcns: Can gcns go as deep as cnns?," in *CVPR*, 2019.

[7] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li, "Simple and deep graph convolutional networks," in *ICML*, 2020.

[8] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2020.

[9] James Bergstra and Yoshua Bengio, "Random search for hyperparameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[10] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[11] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[12] Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse, "Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions," in *ICLR*, 2019.

[13] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al., "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

[14] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu, "Graph neural architecture search," in *IJCAI*, 2020.

[15] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu, "Auto-gnn: Neural architecture search of graph neural networks," *arXiv preprint arXiv:1909.03184*, 2019.

[16] Kwei-Herng Lai, Daochen Zha, Kaixiong Zhou, and Xia Hu, "Policy-gnn: Aggregation optimization for graph neural networks," in *SIGKDD*, 2020.

[17] Xiaodong Jiang, Pengsheng Ji, and Sheng Li, "Censnet: Convolution with edge-node switching in graph neural networks," in *IJCAI*, 2019.

[18] Saed Rezayi, Handong Zhao, Sungchul Kim, Ryan Rossi, Nedim Lipka, and Sheng Li, "EDGE: Enriching knowledge graph embeddings with external text," in *NAACL*, 2021.

[19] Ronghang Zhu, Xiaodong Jiang, Jiasen Lu, and Sheng Li, "Transferable feature learning on graphs across visual domains," in *ICME*, 2021.

[20] Heng-Shiou Sheu and Sheng Li, "Context-aware graph embedding for session-based news recommendation," in *ACM RecSys*, 2020.

[21] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun, "Spectral networks and locally connected networks on graphs," in *ICLR*, 2014.

[22] Matthias Feurer and Frank Hutter, *Hyperparameter Optimization*, pp. 3–33, Springer International Publishing, 2019.

[23] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *ICML*, 2017.

[24] Stefan Falkner, Aaron Klein, and Frank Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *ICML*, 2018.

[25] Zhiqiang Tao, Yaliang Li, Bolin Ding, Ce Zhang, Jingren Zhou, and Yun Fu, "Learning to mutate with hypergradient guided population," in *NeurIPS*, 2020.

[26] Min Shi, David A. Wilson, Xingquan Zhu, Yu Huang, Yuan Zhuang, Jianxun Liu, and Yufei Tang, "Evolutionary architecture search for graph neural networks," *arXiv preprint arXiv:2009.10199*, 2020.

[27] Shengli Jiang and Prasanna Balaprakash, "Graph neural network architecture search for molecular property prediction," *arXiv preprint arXiv:2008.12187*, 2020.

[28] Smriti Bhagat, Graham Cormode, and S Muthukrishnan, "Node classification in social networks," in *Social network data analytics*, pp. 115–148. 2011.

[29] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.