PRELIMINARY VERSION: DO NOT CITE
The AAAI Digital Library will contain the published
version some time after the conference

One-shot Graph Neural Architecture Search with Dynamic Search Space

Yanxi Li[1], **Zean Wen**[1], **Yunhe Wang**[2], **Chang Xu**[1]
[1] School of Computer Science, University of Sydney, Australia
[2] Noah's Ark Lab, Huawei Technologies, China
yali0722@uni.sydney.edu.au, *zwen2780@uni.sydney.edu.au*,
yunhe.wang@huawei.com, *c.xu@sydney.edu.au*

Abstract

Relying on the diverse graph convolution operations that have emerged in recent years, graph neural networks (GNNs) are shown to be powerful to deal with high-dimensional non-Euclidean domains, such as social networks or citation networks. Despite the tremendous human efforts been taken to explore new graph convolution operations, there are a few attempts to automatically search operations in GNNs. The search space of GNNs is significantly larger than that of CNNs, because of diverse components in the message-passing of GNNs. This, therefore, prevents the straightforward application of classical NAS methods for GNNs. In this work, we propose a novel *dynamic one-shot search space* for multi-branch neural architectures of GNNs. The dynamic search space maintains a subset of the large search space along with a set of importance weights for operation candidates in the subset as the architecture parameters. After each iteration, the subset is pruned by removing candidates with low importance weights and is expanded with new operations. The dynamic subsets of operation candidates are not uniform but is individual for each edge in the computation graph of the neural architecture, which can ensure the diversity of operations in the final architecture is as competitive as direct search in the large search space. Our experiments of semi-supervised and supervised node classification on citation networks, including Cora, Citeseer, and Pubmed, demonstrate that our method outperforms the current state-of-the-art manually designed architectures and reaches competitive performance to existing GNN NAS approaches with up to 10 times of speedup.

Introduction

Although standard deep feature learning methods, e.g. Convolutional Neural Networks (CNNs), are successful in tackling grid-like structures (LeCun et al. 1998; Krizhevsky, Sutskever, and Hinton 2012), such as images, videos, and voices, they fail to learn features on high-dimensional non-Euclidean domains, such as molecular structures, social networks, and citation networks, which are typically described by graph structures. To learn graph features from such domains, Graph Neural Networks (GNNs) (Gori, Monfardini, and Scarselli 2005; Scarselli et al. 2008) have emerged. In recent years, GNNs have been demonstrated to be promising to handle graph-structured data, which have increasingly attracted the interest of researchers.

The development of GNNs is a generalization of CNNs from low-dimensional regular grids to high-dimensional irregular grids (Defferrard, Bresson, and Vandergheynst 2016). There are two different categories of approaches to defining a GNN. The spectral-based approaches (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2016; Bianchi et al. 2019) utilize graph Fourier transform and define graph convolutional operators in the spectral domain, and the spatial-based approaches (Veličković et al. 2017; Hamilton, Ying, and Leskovec 2017) define localized message passing rules to aggregate feature information between neighboring nodes.

Architectures of popular GNNs are often manually designed, which requires tremendous prior knowledge and intensive experiments. The automation of neural architecture design has been popular for CNNs and RNNs and is widely known as Neural Architecture Search (NAS). There are a few attempts of applying RL-based NAS approaches to design GNNs. GraphNAS (Gao et al. 2020) and AGNN (Zhou et al. 2019) propose to search for GNNs from the spatial perspective. The message passing process consists of several operators, typically including correlation coefficient calculation and aggregation. The type and hyperparameters for those operators are searched, and a chain-structured network is constructed by stacking obtained message passing layers. This kind of approaches corresponds to macro search in NAS for CNNs (Elsken, Metzen, and Hutter 2018). GraphNAS (Gao et al. 2020) also proposes a micro search approach. In the micro setting, hierarchical computational graphs of fixed operations, e.g. GCN (Kipf and Welling 2016) or GAT (Veličković et al. 2017), is searched as the basic building blocks. Such blocks are then stacked to build a GNN.

However, there are several drawbacks to such RL-based NAS methods. Firstly, RL-based NAS usually needs to build, train, and evaluate various neural architectures individually from scratch, which is extremely time-consuming. A common technique to reduce such search costs in NAS for CNNs is parameter sharing (Pham et al. 2018). However, Zhou et al. (2019) points out that the parameter sharing mechanism for CNNs is unstable for heterogeneous GNNs

and should be constrained. Secondly, in the micro GraphNAS setting, the RL controller is trained to select from only 12 fixed operations and their connection patterns (Gao et al. 2020), which leads to a huge gap of candidates number comparing to the macro setting.

In this paper, we propose a novel one-shot graph NAS with a dynamic search space. We search for micro architectures built with repeated blocks, and we also design the operation candidates from the message passing perspective. This results in 224 message passing operation candidates along with 2 special candidates including none for no connection and $\mathrm{identity}$ for skip-connection, which is around 18 times larger than the GNN search space of the micro GraphNAS (Gao et al. 2020) and is around 28 times larger than the CNN search space of DARTS (Liu, Simonyan, and Yang 2018). In our one-shot setting, a hyper-network containing all possible architectures as sub-graphs, each of which is constructed by the weighted sum of operation candidates. Instead of sharing parameters among operations in different architectures (Zhou et al. 2019), they are trained in parallel. To tackle the challenge of discovering an optimal architecture from such a large number of operations, we turn to maintain a dynamically updated smaller subset of the search space. An architecture parameter is introduced as the importance weights of operations in the subset. The subset of search space is iteratively pruned by removing operations that have low importance weights. Only the top-K operations are retained, and then the subset is expanded with new samples from a queue of inactive operations for the next iteration. The iteration ends when there are no more queued candidates. Our experiments of semi-supervised and supervised node classification on citation networks, including Cora, Citeseer, and Pubmed, demonstrate that our method outperforms the current state-of-the-art manually designed architectures and reaches competitive performance to existing GNN NAS approaches with up to 10 times of speedup.

Related Works

Graph Neural Networks

A great number of GNN architectures have been designed by researchers, accompanied by their superior performance in molecules, bioinformatics, social network, and computer vision graph-structured data processing (Kipf and Welling 2016; Wu et al. 2019; Morris et al. 2019; Li et al. 2019). Graph Convolutional Network (GCN) introduces a layer-wise propagation rule, follow which each node's feature is updated by aggregating the information from its neighbors (Kipf and Welling 2016). Based on this GCN method, a simplified version – Simple Graph Convolution (SGC) – was proposed to reduce the excess complexity of GCNs by repeatedly removing the nonlinearities between GCN layers and collapsing the resulting function into a single linear transformation (Wu et al. 2019).

One of the inherent limitations of current GNNs identified by many researchers is related to the feature aggregation scheme (Morris et al. 2019; Klicpera, Bojchevski, and Günnemann 2018; Hu et al. 2019). (Morris et al. 2019) demonstrated that the standard GCN can be considered as the neural version of the 1-dimensional Weiseriler-Leman (WL) algorithm, of which the expressive power is limited because of the fixed feature construction scheme. K-GNNs (GraphGCN), based on the k-dimensional WL algorithm, was proposed to exceed this performance limit. (Klicpera, Bojchevski, and Günnemann 2018) indicated that current aggregation schemes extract only limited neighbor information. Approximated Personalized Propagation of Neural Prediction (APPNP) utilizes the personalized PageRank aggregation scheme (Page et al. 1999) and adds a teleporting to the root node, enabling the node to leverage the information from a larger neighborhood. To address the same limited receptive field problem, Hierarchical GCN (H-GCN) constructs a coarsening mechanism to increase the receptive field and capture the global information. (Hu et al. 2019). (Bianchi et al. 2019) proposed a novel ARMA filter to overcome the shortage of the original polynomial filter which are sensitive to the graph signal and structure.

Neural Architecture Search

Neural Architecture Search (NAS), a burgeoning automatic architecture engineering technique, can construct a complex and high-performance neural network architecture without human engaged trial and error. Early NAS algorithms focus on the *macro-architecture* search manner. Specifically, they aim to optimize operations together with their hyper-parameters in different layers of a chain-structured neural network simultaneously. Motivated by the repeated motifs which are the essential components in the high-performing hand-crafted CNN architecture, NASNet (Zoph et al. 2018) proposed to search for normal and reduction cells (*micro-architecture*) as the building blocks for the final neural network. Searching on cells instead of the whole architecture can dramatically reduce the size of searching space, accelerating the searching procedure, while proved to have a better performance (Elsken, Metzen, and Hutter 2018; Zoph et al. 2018).

Early NAS methods usually incorporate reinforcement learning (RL) into the searching strategy of NAS (Baker et al. 2016; Zoph and Le 2016; Zhong et al. 2018; Zoph et al. 2018). In the RL-based NAS approaches, the construction of neural architectures is considered as a Markov decision process (MDP), and an RL agent is trained to learn the generation of architectures. The reward for the agent is based on the estimation of the architecture performance (Elsken, Metzen, and Hutter 2018). RL-based NAS methods repeatedly train discrete architectures from scratch for evaluation, which is time-consuming.

To reduce the search cost, differentiable NAS approaches (Liu, Simonyan, and Yang 2018; Dong and Yang 2019) utilize a continuous relaxation of the architecture representation, allowing to optimize the architecture by using gradient descend. This strategy outperforms RL-based NAS approaches with less computation cost (several GPU hours comparing to thousands of GPU days). Besides direct reducing the training cost, CARS (Yang et al. 2020) proposes a novel efficient continuous evolutionary approach based on the historical evaluation. Similarly, PVLL-NAS (Li et al. 2020a) schedules their evaluation with a performance esti-

mator, who samples neural architectures for both architecture searching and iterative training of the estimator itself. Another direction benefits from the efficiency of domain adaptation. AdaptNAS (Li et al. 2020b) proposes to adapt neural architectures searched on a small proxy task to a large target task, which reduces the search cost and guarantees the performance on large-scale tasks in the meantime.

NAS can help to solve the limitations of GNN architectures addressed above. The automatic architecture search solves the labor-intensive problem of GNN architecture design. Besides, by assembling a more complex topological search space, the searched architecture can be more complex and robust to different graph data. There are some finished works of neural architecture search on GNN architecture such as Auto-GNN (Zhou et al. 2019) and GraphNAS (Gao et al. 2020). They used the reinforcement learning technique to perform the neural architecture search and achieve state-of-art performances. In this paper, we proposed a GNN-based architecture search algorithm, based on Differentiable Architecture Search (Liu, Simonyan, and Yang 2018). Our experiments showed that our results are competitive with that of the current NAS GNN while our algorithm requires fewer computation hours.

Methodology

In this section, we first formally define the formulation of NAS for GNN from a bi-level optimization perspective. Then, we define the search space for GNNs along with a set of operation candidates in a message-passing form. Finally, we discuss how to perform one-shot NAS with the large-scale operation candidates by utilizing a dynamic search space.

Problem Formulation

Let $\mathcal{A}$ be a search space consisting of neural architectures $\boldsymbol{\alpha}$. Neural architecture search (NAS) is to find an optimal architecture $\boldsymbol{\alpha}^* \in \mathcal{A}$ that maximizes or minimizes the given performance metrics M. A widely used performance metric could be the validation loss $\mathcal{L}_{valid}$ on a held-out validation set $\mathcal{D}_{valid}$, and the corresponding network weights are the optimal weights $\boldsymbol{w}^*$ obtained on a training set $\mathcal{D}_{train}$, which minimizes the training loss $\mathcal{L}_{train}$. In general, NAS can be formulated as a bi-level optimization problem:

$$\min_{\boldsymbol{\alpha}} \quad \mathcal{L}_{valid}(\boldsymbol{w}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}) \tag{1}$$

$$\textbf{s.t.} \quad \boldsymbol{w}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{w}} \mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\alpha}), \tag{2}$$

where Eq. 1 is the upper-level problem w.r.t. the neural architecture $\boldsymbol{\alpha}$, and Eq. 2 is the lower-level problem of the network weight $\boldsymbol{w}$. These two levels of the NAS problem can be optimized alternately until convergence or reaching a maximum iteration number.

Search Space

We use a hierarchical architecture with branches for the building blocks, which is a computational graph represented by a directed acyclic graph (DAG). In each computational graph, there are two input nodes, one or more intermediate nodes, and an output node. Let there be N nodes in the computational graph, i.e. $nodes = \{\boldsymbol{I}^{(i)} | 0 \leq i \leq N-1\}$. Then, $\boldsymbol{I}^{(0)}$ and $\boldsymbol{I}^{(1)}$ are input nodes corresponding to the previous two cells $\boldsymbol{h}_{c-2}$ and $\boldsymbol{h}_{c-1}$, $\boldsymbol{I}^{(N-1)}$ is the output node of the current cell corresponding to $\boldsymbol{h}_c$, and other $N-3$ nodes are intermediate nodes. The directed edges (i, j) of the computational graph correspond to an operation $o^{(i,j)}(\cdot)$. For example, Figure 1 shows the case where $N = 5$.

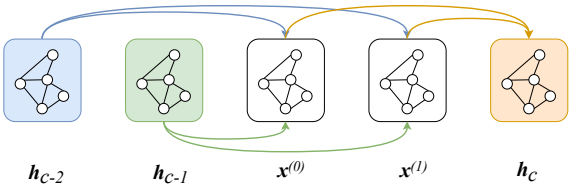


Figure 1: Architecture search space when $N = 5$.

To calculate the intermediate nodes, we take the sum of all the edges that direct to it:

$$\boldsymbol{I}^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right), 2 \leq j \leq N-2. \tag{3}$$

where $o_{i,j} \in \mathcal{O}$ is an operation, and $\mathcal{O}$ is a set of operation candidates. The output is the concatenation of all the intermediate nodes

$$\boldsymbol{I}^{(N-1)} = \Big\|_{i=2}^{N-2} \boldsymbol{I}^{(i)}. \tag{4}$$

The set $\mathcal{O}$ defines the possible operations that can be used in the architecture. Instead of using a few predefined fixed candidates for $o \in \mathcal{O}$, the operations are defined in a message-passing form, where feature information is aggregated between neighboring nodes. The output of node v in layer l is calculated by

$$h_v^{(l)} = \text{Aggregate}\left(\left\{e_{u,v}^{(l)} W^{(l)} h_u^{(l-1)} : u \in N(v)\right\}\right), \tag{5}$$

where $\text{Aggregate}(\cdot)$ is an aggregation function, $e_{u,v}^{(l)}$ is a correlation coefficient, $W^{(l)}$ is the network weight, $h_u^{(l-1)}$ is the output of previous layer (or the input feature x_u if $l = 1$), and $N(v)$ is the receptive field of the node v.

The reception field $N(v)$ defines how to sample local neighbors for aggregation (Hamilton, Ying, and Leskovec 2017). We use a fixed setting and sample all the first order neighbors of the node v as the reception field. The aggregation function can be any permutation invariant operations, such as sum, mean and max. We also include a multilayer perceptron (MLP) based aggregation function proposed by (Xu et al. 2018), which first applies a sum aggregation and then passes the sum result through an MLP. The correlation coefficient is calculated between the current node v and each node u in its reception field $N(v)$. The formulas of various correlation coefficients are described in Table 1. We further define an activation function $\text{Act}(\cdot)$ for the output. All the components and their corresponding options are listed in Table 2.

Message Passing	Formula
Gonst	$e_{uv}^{\text{Gonst}} = 1$
GCN	$e_{uv}^{\text{GCN}} = 1/\sqrt{d_u d_v}$
GAT	$e_{uv}^{\text{GAT}} = \text{leaky\_relu}(W_l * h_u + W_r * h_v)$
Sym-GAT	$e_{uv}^{\text{Sym}} = e_{uv}^{\text{GAT}} + e_{vu}^{\text{GAT}}$
Cos	$e_{uv}^{\text{Cos}} =< W_l * h_u, W_r * h_v >$
Linear	$e_{uv}^{\text{Linear}} = \tanh(\text{sum}(W_l * h_u))$
Gene-Linear	$e_{uv}^{\text{Gene}} = W_a * \tanh(W_l * h_u + W_r * h_v)$

Table 1: Formulas of various correlation coefficients.

Component	Options
$e_{u,v}$	Gonst, GCN, GAT, Sym-GAT, Cos, Linear, Gene-Linear
$\text{Aggregate}(\cdot)$	sum, mean, max, MLP
$\text{Act}(\cdot)$	sigmoid, tanh, relu, leaky_relu, relu6, elu, linear, softplus

Table 2: The components of message-passing based operations and their options.

One-shot Graph NAS

The repeatedly training of various neural architectures from scratch for evaluation is time-consuming, and the constrained parameter sharing mechanism (Zhou et al. 2019) is limited. In the constrained parameter sharing, the parameters can be shared if and only if the ancestor and the offspring have the same shape of input and output, use the same kind of correlation coefficient function and activation function, and is not a batch normalization layer or a skip connection. Parameters violate such constraints still need to be trained from scratch. Besides, extra computation complexity is introduced comparing to the traditional strategy to check the constraints for each pair of parameters to be shared, which is, unfortunately, CPU intensive and cannot be accelerated by GPUs.

Although the explicit parameter sharing is limited, there is a technique to share parameters implicitly among architectures, which is the one-shot search method. The one-shot search method has been widely adopted for convolutional neural architecture search. It successfully reduces the search cost of CNN architectures from thousands of GPU days (Zoph et al. 2018) to several GPU hours (Liu, Simonyan, and Yang 2018; Dong and Yang 2019). In one-shot NAS, a hyper-network containing all possible architectures in the search space as its sub-graphs is constructed and trained once for evaluation. Operation candidates are weighted summed according to an architecture parameter $\boldsymbol{\alpha}$ in the hyper-network, which reformulates NAS problems to the learning of the importance weights $\boldsymbol{\alpha}$ for candidates. After search, discrete architectures can be extracted from the hyper-network.

Algorithm 1 Search with dynamic search space

Input: a set of operation candidates $\mathcal{O}$, the number of nodes N in a cell, the maximum size M of the candidates subset, the maximum epoch max_epoch of inner-loop to optimize a hyper-network, the number K of top candidates to be remained after each iteration

Output: architecture parameter $\boldsymbol{\alpha}$

```
Random sample O_{i,j} ∈ O of size M
while O ≠ ∅ do
    Let O ← O − O_{i,j}
    Random initialize α^{(i,j)}
    for epoch ∈ {1, ..., max_epoch} do
        Update weights w with L_train(w, α)
        Update architecture α with L_valid(w, α)
    end for
    Select top K operations O*_{i,j} ∈ O_{i,j}
    Random sample O'_{i,j} ∈ O of size M − K
    Let O_{i,j} ← O*_{i,j} ∪ O'_{i,j}
end while
```

The remaining problem is how to build a one-shot with the large scale search space. According to the options listed in Table 2, there are 224 different combinations of components. Besides, we also use two fixed operations including a none representing not connected and an identity represent skip connection. This makes the number of operation candidates too large to be trained in parallel in a weighted summed one-shot model due to its large demand for GPU memory. An existing solution to this issue in NAS of CNN is sampling and training one path per epoch (Dong and Yang 2019). However, we argue that when the operation number is too large (226 compared to 8 in the previous CNN work), the probability of an individual operation to be sampled is too low to train it properly. Especially when the number of epochs is less than the number of operation candidates, there must be operations that have never been sampled.

To solve this problem, we propose a dynamic search space, where a small subset of operation candidates $\mathcal{O}_{i,j} \in \mathcal{O}$ is maintained for each directed computation edge (i, j) in the architecture such that $j \in \{2, \ldots, N-2\}$ and $i \in \{0, \ldots, j-1\}$. At the beginning of search, we first randomly initialize $\mathcal{O}_{i,j}$. During search, the subset is then updated iteratively. The operation candidates are ranked according to their corresponding architecture parameter $\boldsymbol{\alpha}_o^{(i,j)}$, and only the top-K operations are kept after an iteration. The basic idea is that if an operation is ranked lower than the other in a subset, it is ranked lower than the others in the universe. Therefore, operations with low $\boldsymbol{\alpha}_o^{(i,j)}$ can be replaced with new operations sampled from the remaining candidates for the next iteration. Note that the dynamic subsets of operation candidates are not uniform for all the edges in the cell but are maintained individually for each edge, which ensures the diversity of operations in the final architecture.

To construct a one-shot hyper-network of the dynamic search space, the architecture parameter $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}^{(i,j)}\}$ is

		Cora	Citeseer	Pubmed
#Nodes		2708	3327	19717
#Edges		5429	4732	44338
#Features		1433	3703	500
#Classes		7	6	3
Semi	#Train Nodes	140	120	60
	#Valid Nodes	500	500	500
	#Test Nodes	1000	1000	1000
Full	#Train Nodes	1708	2327	18717
	#Valid Nodes	500	500	500
	#Test Nodes	500	500	500

Table 3: Statistics and splits of Cora, Citeseer, and Pubmed.

utilized. The mixed operation on edge (i, j) is calculated according to the softmax over $\boldsymbol{\alpha}^{(i,j)}$ by:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}_{i,j}} \frac{\exp\left(\boldsymbol{\alpha}_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}_{i,j}} \exp\left(\boldsymbol{\alpha}_{o'}^{(i,j)}\right)} o(x), \quad (6)$$

where $\bar{o}^{(i,j)}$ is the weighted mixed operation on edge (i, j) in the hyper-network. Therefore, the hyper-network is parameterized by two factors, the network weights $\boldsymbol{w}$ and the architecture parameter $\boldsymbol{\alpha}$. To evaluate and optimize the hyper-network, we use a commonly used loss function for classification tasks, negative log likelihood (NLL):

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[-y \log \left(f(\boldsymbol{x}|\boldsymbol{w}, \boldsymbol{\alpha})\right)\right]. \quad (7)$$

Following the bi-level optimization setting in Eqs. 1 and 2, the architecture parameter is update with the gradient of validation loss:

$$\boldsymbol{\alpha}' \leftarrow \boldsymbol{\alpha} - \eta_{\boldsymbol{\alpha}} \cdot \nabla_{\boldsymbol{\alpha}} \mathcal{L}_{valid}(\boldsymbol{w}, \boldsymbol{\alpha}), \quad (8)$$

where $\eta_{\boldsymbol{\alpha}}$ is the architecture learning rate, and $\mathcal{L}_{valid}(\boldsymbol{w}, \boldsymbol{\alpha})$ is the validation loss calculated on a held-out validation set $\mathcal{D}_{valid}$.

After the architecture parameter is optimized in the one-shot hyper-network, top-K operation candidates on each edge (i, j) are selected given the $\boldsymbol{\alpha}^{(i,j)}$:

$$\mathcal{O}_{i,j}^{*} = \mathrm{TOP}_K(\mathcal{O}_{i,j} \mid \boldsymbol{\alpha}^{(i,j)}). \quad (9)$$

For each edge (i, j), we also maintain a queue of unsampled operations. The top-K operations are then combined with a set of newly sampled architecture from the queue of unsampled operations, and the inner-loop starts again. The overall procedure is as shown in Algorithm 1. There is an inner-loop, where a sampled subset of operation candidates $\mathcal{O}_{i,j} \in \mathcal{O}$ for each edge is searched with an one-shot hyper-network, and an outer-loop, where the subset of operation candidates is dynamically updated according to the current architecture parameter $\boldsymbol{\alpha}_{i,j}$ for the edge.

Experiments

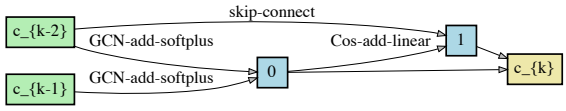
Datasets

We search for architectures and test their performances on citation networks for node classification, including Cora, Citeseer, and Pubmed. Following the splits used by Yang, Cohen, and Salakhudinov (2016) and Gao et al. (2020), for the semi-supervised learning, we use 20 nodes per class for training and use 500 and 1,000 nodes for validation and testing, respectively For the full supervised learning, we follow the splits of Gao et al. (2020), where 500 nodes are used for validation, 500 nodes are used for testing, and all the remaining nodes are used for training. The detailed statistics of the dataset along with the splits are shown in Table 3.

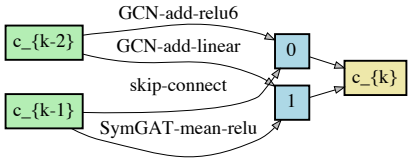
Baseline Methods

In order to demonstrate the effectiveness of our NAS method, we compare the found architectures with various categories of the state-of-the-art architectures, including manually crafted architectures and various NAS searched architectures.

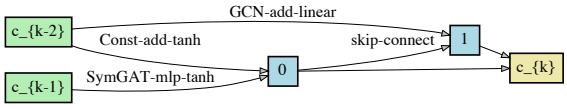
- **Manually crafted architectures:** Firstly, we consider two architectures that includes in our search space: GCN (Kipf and Welling 2016), which is equivalent to "GCN-sum-relu", and GAT (Veličković et al. 2017), which is equivalent to "GAT-sum-elu". Besides, we also compare with several newly published and complicated methods: APPNP (Klicpera, Bojchevski, and Günnemann 2018), which utilize an improved propagation scheme based on personalized PageRank, ARMA (Bianchi et al. 2019), which utilizes auto-regressive moving average (ARMA) filters instead of the polynomial ones, and H-GCN (Hu et al. 2019), which learns hierarchical representations for graphs.



(a) The best architecture on Cora;



(b) The best architecture on Citeseer;



(c) The best architecture on Pubmed;

Figure 2: The best searched architectures on the semi-supervised tasks.

Type	Model	Cora		Citeseer		Pubmed	
		semi	full	semi	full	semi	full
Manually Crafted	GCN (Kipf and Welling 2016)	$81.4 \pm 0.5\%$	$90.2 \pm 0.0\%$	$70.9 \pm 0.5\%$	$80.0 \pm 0.3\%$	$79.0 \pm 0.4\%$	$87.8 \pm 0.2\%$
	GAT (Veličković et al. 2017)	$83.0 \pm 0.7\%$	$89.5 \pm 0.3\%$	$72.5 \pm 0.7\%$	$78.6 \pm 0.3\%$	$79.0 \pm 0.3\%$	$86.5 \pm 0.6\%$
	ARMA (Bianchi et al. 2019)	$82.8 \pm 0.6\%$	$89.8 \pm 0.1\%$	$72.3 \pm 1.1\%$	$79.9 \pm 0.6\%$	$78.8 \pm 0.3\%$	$88.1 \pm 0.2\%$
	APPNP (Klicpera et al. 2018)	$83.3 \pm 0.1\%$	$90.4 \pm 0.2\%$	$71.8 \pm 0.4\%$	$79.2 \pm 0.4\%$	$80.2 \pm 0.2\%$	$87.4 \pm 0.3\%$
	H-GCN (Hu et al. 2019)	$79.8 \pm 1.2\%$	$89.7 \pm 0.4\%$	$70.0 \pm 1.3\%$	$79.2 \pm 0.5\%$	$78.4 \pm 0.6\%$	$88.0 \pm 0.5\%$
Macro NAS	AGNN (Zhou et al. 2019)	$83.6 \pm 0.3\%$	-	$73.8 \pm 0.7\%$	-	$79.7 \pm 0.4\%$	-
	GraphNAS (Gao et al. 2020)	$83.7 \pm 0.4\%$ (2 GPU Hrs)	-	$73.5 \pm 0.3\%$ (2 GPU Hrs)	-	$80.5 \pm 0.3\%$ (9 GPU Hrs)	-
Micro NAS	GraphNAS (Gao et al. 2020)	-	$90.6 \pm 0.3\%$ (6 GPU Hrs)	-	$81.2 \pm 0.5\%$ (6 GPU Hrs)	-	$91.2 \pm 0.3\%$ (12 GPU Hrs)
	DSS (Ours)	$83.9 \pm 0.3\%$	$91.0 \pm 0.2\%$ (0.9 GPU Hrs)	$73.3 \pm 0.3\%$	$81.4 \pm 0.4\%$ (0.8 GPU Hrs)	$80.3 \pm 0.2\%$	$88.2 \pm 0.4\%$ (0.9 GPU Hrs)

Table 4: Test accuracy on node classification tasks.

- **Macro NAS architectures:** Both GraphNAS (Gao et al. 2020) on the semi-supervised tasks and AGNN (Zhou et al. 2019) searched for individual operations in different chain-structured layers of the entire network along with their hyper-parameters, such as dimensions and number of attention heads. These architectures are typically referred to as macro architectures (Elsken, Metzen, and Hutter 2018).
- **Micro NAS architectures:** GraphNAS (Gao et al. 2020) searches micro architectures consisting of a hierarchical multi-branch structure of 12 predefined operations on the full-supervised tasks.

Results on Node Classification

We search for neural architectures with the semi-supervised setting on each dataset. After search, we retrain the searched architectures in both the semi- and full- supervised setting. To construct the hyper-networks, we allow 2 intermediate nodes, which correspond to 5 directed computation edges in the hyper-networks and 4 directed computation edges in the final discrete architectures. The hyper-network has 16 channels and 2 layers. We follow Kipf and Welling (2016) and use a single GCN instead of MLP as the classifier. Both the network weights and the architecture parameters are optimized with ADAM. The network learning rate is set to 0.007, and the architecture learning rate is set to 0.1. The hyper-network is trained for 100 epochs. Dropout and weight decay are applied as regularization. We use a dropout rate of 60%, and use 3e-4 for the weight decay of network weights and 1e-3 for the weight decay of architecture parameters. The GCN classifier is excluded from weight decay. For our dynamic search space, we use 224 message passing operations and 2 special operations as aforementioned. The size M of the dynamic subset of operation candidates is 10. When the subset is updated, top-3 operations are kept.

After search with the one-shot hyper-network, to derive a discrete architecture, we select the strongest operation on the top-2 strongest edges. For example, if a node in a computational graph has 3 input edges, we first select the top-1 operation o^* on each edge according to $\alpha_{i,j}$. Then, we compare edges according to the architecture parameter of each selected operation $\alpha_{o^*}^{(i,j)}$ and retain the top-2 edges. The searched architectures on each dataset are demonstrated in Figure 2.

The obtained architectures are evaluated by retraining. For retraining, we still use networks with 2 layers and optimize them with ADAM. The other hyper-parameters are tuned over the following options:

- Hidden channels: $\{16, 64, 128, 256\}$;
- Learning rate: $\{0.05, 0.005, 0.001, 0.0001\}$;
- Dropout rate: $\{0.0, 0.3, 0.6, 0.9\}$;
- Weight decay: $\{1e-4, 5e-5\}$.

The test accuracy of searched architectures are demonstrated in Table 4. Compared to the semi-supervised setting of GraphNAS on Cora and Citeseer, our proposed one-shot method achieves twice speedup, while on Pubmed it achieves 10 times of speedup. Because the manually crafted GNNs use the same architecture for both semi- and full-supervised tasks, we transfer the architectures searched with semi-supervision to full-supervision without any modification.

On semi-supervised tasks, our architectures consistently outperform those hand-crafted ones and can reach competitive performance to AGNN and GraphNAS with less search cost than GraphNAS (detailed search cost is not reported in AGNN, but they claim a 12-GPU-hours search by their method without parameter sharing). The full-supervised performance of our architectures is slightly lower than GraphNAS on Pubmed. This might because they search for different architectures that are specific to full-supervised tasks, while we directly transfer architectures searched with semi-supervision. However, our architectures can still consistently outperform all the manually crafted ones under full-supervision.

Search with Dynamic Search Space

To show the effectiveness of using dynamic search space, we compare our method with a fixed search space and a random search space. To construct a fixed search space, we evaluated all the operation candidates individually Chain-structured networks with 2 layers and 64 channels are constructed and evaluated. To ensure the diversity of the search space, we select the top aggregation function and activation function for each correlation coefficient. The test accuracy of top operations are shown in Table 5. For random search space, we randomly sample the subset for each edge (i, j) and search for only 1 iteration without the top-K updating. We repeat the random initialization on each dataset for 3 times to get different random search space.

Operation	**Cora**
Const-mean-elu	77.4%
GCN-add-softplus	79.7%
GAT-add-tanh	79.6%
SymGAT-add-sigmoid	80.7%
Cos-add-linear	73.0%
GeneLinear-mlp-elu	81.3%
Linear-add-tanh	80.5%

Table 5: Top operations used for fix search space. Accuracy is tested on Cora with the semi-supervised setting.

The test accuracy of search architectures from each search space on the semi-supervised tasks are reported in Table 6. Not surprisingly, the proposed dynamic search space consistently reaches the best performance among all the three search spaces. However, things become interesting when we compare the fixed search space and the random search space. The architectures searched in the fixed search space reach better performance on Cora and Citeseer than the random search space, but are outperformed by the latter on Pubmed. Along with the superior performances of dynamic search space, this demonstrates that operations perform well when they are solely used do not always perform that well when they are used in a complex structure. Besides, randomly sampled candidates also have a chance to outperform candidates perform well when they are solely used, although it is highly unstable due to its random nature.

Search Space	**Cora**	**Citeseer**	**Pubmed**
Fixed	$83.4 \pm 0.2\%$	$71.8 \pm 0.2\%$	$78.9 \pm 0.3\%$
Random	$82.7 \pm 0.9\%$	$71.2 \pm 0.2\%$	$79.2 \pm 0.2\%$
Dynamic	$83.9 \pm 0.3\%$	$72.2 \pm 0.3\%$	$80.3 \pm 0.2\%$

Table 6: Semi-supervised test accuracy of architectures searched on different search space. The random search spaces are sampled for 3 times on each dataset to get different initialization.

Dynamic Search Space Updating

There are two hyper-parameters of the dynamic search space, i.e. the subset size M of dynamic operation candidates and the K for top-K operation selection. They impact the search cost and final performance together. In terms of search cost, the smaller M or the larger K will lead to more iterations and longer GPU hours. The number of iterations can be calculated by

$$iterations = 1 + \left\lceil \frac{|\mathcal{O}| - M}{M - K - 2} \right\rceil, \tag{10}$$

where $|\mathcal{O}|$ is the total number of operation candidates, and the constant 2 corresponding to the 2 special operations. Although increase M can reduce the total number of iterations, a large M requires too much GPU memory and causes overflow. We empirically set the upper bound of M to 10.

Subset Size	**Top-K**	**GPU Hours**	**Cora** semi	**Cora** full
8	1	0.8	$82.2 \pm 0.3\%$	$91.0 \pm 0.2\%$
	3	1.2	$82.8 \pm 0.3\%$	$91.1 \pm 0.1\%$
10	1	0.7	$82.6 \pm 0.4\%$	$90.8 \pm 0.3\%$
	3	0.9	$83.9 \pm 0.3\%$	$91.0 \pm 0.2\%$

Table 7: Different hyper-parameters for the dynamic search space.

The results are as shown in Table 6. We set M to our empirical upper-bound 10 and a smaller value 8. The results show that decreasing M indeed increases the GPU hours around 0.1 to 0.3 depending on K without any improvement in accuracy. For K, we only retain no more than 1 operation on each edge after search according to our deriving rule, which means $K = 1$ is sufficient enough in theory. However, to avoid instability in practice which might cause the miss omitting of good operations, we set $K = 3$ and give the second and third best operations "one more chance". The results show that a larger K can ensure better results.

Conclusion

In this paper, we propose a novel framework for one-shot graph neural architecture search with a dynamic search space. The proposed dynamic search space solves the challenges of applying one-shot hyper-networks under a large scale search space and meanwhile ensures the performance and diversity of final searched architectures. Instead of sharing parameters, the one-shot hyper-networks train and evaluate various architectures in parallel, which avoids the limitation of parameter sharing in GNNs on input and output shapes and operation types. Extensive experiments demonstrate that our method outperforms the current state-of-the-art manually designed architectures and reaches competitive performance to existing GNN NAS approaches with up to 10 times of speedup.

Acknowledgement

This work was supported in part by the Australian Research Council under Projects DE180101438 and DP210101859.

References

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167* .

Bianchi, F. M.; Grattarola, D.; Alippi, C.; and Livi, L. 2019. Graph neural networks with convolutional arma filters. *arXiv preprint arXiv:1901.01343* .

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 3844–3852.

Dong, X.; and Yang, Y. 2019. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, 1761–1770.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* .

Gao, Y.; Yang, H.; Zhang, P.; Zhou, C.; and Hu, Y. 2020. Graph neural architecture search. In *IJCAI*, volume 20, 1403–1409.

Gori, M.; Monfardini, G.; and Scarselli, F. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, 729–734. IEEE.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, 1024–1034.

Hu, F.; Zhu, Y.; Wu, S.; Wang, L.; and Tan, T. 2019. Hierarchical graph convolutional networks for semi-supervised node classification. *arXiv preprint arXiv:1902.06667* .

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* .

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* .

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11): 2278–2324.

Li, G.; Muller, M.; Thabet, A.; and Ghanem, B. 2019. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, 9267–9276.

Li, Y.; Dong, M.; Wang, Y.; and Xu, C. 2020a. Neural architecture search in a proxy validation loss landscape. In *International Conference on Machine Learning*.

Li, Y.; Wang, Y.; Xu, C.; et al. 2020b. Adapting neural architectures between domains. *Advances in Neural Information Processing Systems* 33.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* .

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4602–4609.

Page, L.; Brin, S.; Motwani, R.; and Winograd, T. 1999. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* .

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1): 61–80.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* .

Wu, F.; Zhang, T.; Souza Jr, A. H. d.; Fifty, C.; Yu, T.; and Weinberger, K. Q. 2019. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153* .

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* .

Yang, Z.; Cohen, W.; and Salakhudinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, 40–48. PMLR.

Yang, Z.; Wang, Y.; Chen, X.; Shi, B.; Xu, C.; Xu, C.; Tian, Q.; and Xu, C. 2020. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1829–1838.

Zhong, Z.; Yan, J.; Wu, W.; Shao, J.; and Liu, C.-L. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2423–2432.

Zhou, K.; Song, Q.; Huang, X.; and Hu, X. 2019. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184* .

Zoph, B.; and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* .

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697–8710.