Do Not Train It: A Linear Neural Architecture Search of Graph Neural Networks

Peng Xu[1*], Lin Zhang[2*], Xuanzhou Liu[3], Jiaqi Sun[3], Yue Zhao[4], Haiqin Yang[2], Bei Yu[1]

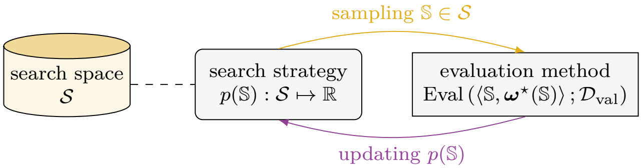
[1]The Chinese University of Hong Kong
[2]International Digital Economy Academy
[3]Tsinghua University
[4]Carnegie Mellon University

Outline

Neural Architecture Search(NAS)

Neural architecture search (NAS) is a technique for automating the design of artificial neural networks. NAS generally includes three major parts[1]:

- Search Space: The search space define the type(s) of ANN that can be designed and optimized.
- Search Strategy: The search strategy defines the approach used to explore the search space.
- Performance Estimation Strategy: The performance estimation strategy evaluates the performance of a possible ANN from its design.


sampling $\mathbb{S} \in \mathcal{S}$
search space $\mathcal{S}$
search strategy $p(\mathbb{S}) : \mathcal{S} \mapsto \mathbb{R}$
evaluation method $\text{Eval}\left(\langle \mathbb{S}, \boldsymbol{\omega}^{\star}(\mathbb{S})\rangle ; \mathcal{D}_{\text{val}}\right)$
updating $p(\mathbb{S})$

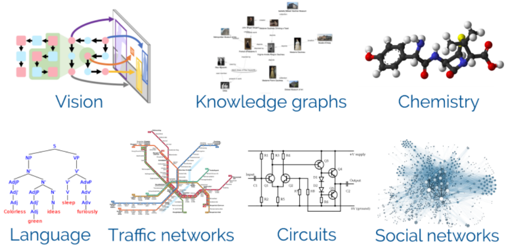
The three components of NAS.

[1]Lingxi Xie et al. (2021). "Weight-sharing neural architecture search: A battle to shrink the optimization gap". In: *ACM Computing Surveys* 54.9, pp. 1–37.

Graph Neural Networks(NAS-GNN)

Graph Neural Networks (GNN) have been a very hot topic in recent years[2].

- Representation learning in graphs
- Define "convolution" on graph(non-grid) data
- Powerful expression capability in dealing with graph structure data
- Applications of Graph Neural Network:
  - Recommendation
  - Fraud Detection
  - Bioinformatics
  - Netlist Representation
  - ...



Typical Graph Structural data.

Applying NAS techniques to the field of GNN to search for data-specific GNN network structures is a natural choice.

[2]Zonghan Wu et al. (2021). "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1, pp. 4–24.

NAS in Graph Neural Networks(NAS-GNN)

NAS in Graph Neural Networks

- Sample-based
  - Prohibitive computational costs

  1. Reinforcement Learning-based: GraphNAS[3]
  2. Evolutionary Algorithm-based: Genetic-GNN[4]
- Weight-Sharing-based
  - Reduce the search effort by reusing the neural weights

  1. **Differential-based**: SANE[5]

[3]Yang Gao et al. (2020). "Graph Neural Architecture Search.". In: *Proc. IJCAI*.
[4]Min Shi et al. (2022). "Genetic-gnn: evolutionary architecture search for graph neural networks". In: *Knowledge-Based Systems* 247, p. 108752.
[5]ZHAO Huan, YAO Quanming, and TU Weiwei (2021). "Search to aggregate neighborhood for graph neural network". In: *Proc. ICDE*, pp. 552–563.

Differential-based NAS-GNN(Darts-GNN)

The differential-based method addresses *two* subproblems of weights and architectures alternatively using a bi-level optimization framework to reduce the search effort[6]:

$$\begin{cases} \alpha^* = \underset{\alpha}{\operatorname{argmax}} \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right), \\ w^* = \underset{w}{\operatorname{argmin}} \mathcal{L}_{train}(\alpha, w) \end{cases} \tag{1}$$

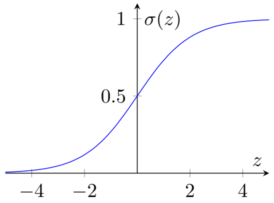
where $\mathcal{L}$ denotes a loss function (e.g., cross-entropy loss) on training and validating dataset w.r.t w, and α is the architecture parameter.

[6] ZHAO Huan, YAO Quanming, and TU Weiwei (2021). "Search to aggregate neighborhood for graph neural network". In: *Proc. ICDE*, pp. 552–563.

Existing Problems

Two optimization challenges cause the **instability** of the differential-based NAS-GNN methods:

- **Optimization challenge due to BLO**: It is difficult to reach optimal state for objectives in the Eq. 1 simultaneously under the bi-level optimization framework. The training process is hard to converge and suffers from oscillations.
- **Inaccurate estimation**: The softmax operation will keep the unimportant architecture parameter as non-zero values.

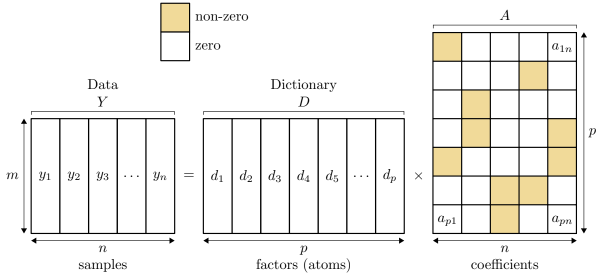


Softmax

Motivation

In this work, we attempt to address the optimization difficulty in the Darts-GNN problem from a new perspective.

- **No-update Scheme**: We utilize the power of untrained GNN[7] in the NAS-GNN problem and provide a theoretical analysis.
- **Sparse Coding**: The ultimate goal of NAS is to assign the coefficients of unimportant operators as zeros.



Sparse Coding

[7] Thomas N. Kipf and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *Proc. ICLR*.

Challenges

We ask two key questions

- *Is updating the GNN weights necessary?*
    - => Do untrained orthogonal weights can lead to optimal architecture search?
- *Can we formulate the NAS-GNN problem as a sparse coding problem?*
    - => Can we find the dictionary for the NAS-GNN problem?

Analysis of No-update Scheme in GNNs

In our paper, we theoretically investigate why an untrained GNN model can attain the same performance as the optimal one.

Theorem

Assume $\mathbf{W}_l(0)$ *is randomly initialized for all* $l \in [1, L]$, *if* $\prod_{l=1}^{L} \mathbf{W}_l(0)$ *is full rank, there must exist a weight matrix for the output layer, i.e.,* $\tilde{\mathbf{W}}_o$, *that makes the final output the same as the one from a well-trained network:*

$$\mathbf{A}^L \mathbf{X} \prod_{l=1}^{L} \mathbf{W}_l^* \mathbf{W}_o^* = \mathbf{A}^L \mathbf{X} \prod_{l=1}^{L} \mathbf{W}_l(0) \tilde{\mathbf{W}}_o. \tag{2}$$

We proved that one can approximate the optimal output by using the orthogonal initialization, which justifies the power of untrained GNN.

Architecture Searching via Sparse Coding

In this work, we treat the entire NAS search space of GNNs as a general GNN model where each layer is a mixture of multiple operators. Muhammet Balcilar et al.[8] proposed that each GNN layer can have a unified expression.

Table: Summary of the studied GNN models.

	Design	Support Type	Comolution Malrix	Frequency Response
MLP	Spectral	Fixed	$C = I$	$\Phi(\boldsymbol{A}) = \mathbf{1}$
GCN	Spatial	Fixed	$C = \tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}$	$\Phi(\bar{\lambda}) = \mathbf{1} - \boldsymbol{\lambda}\bar{p}/(\bar{p}+1)$
GIN	Spatial	Trainable	$C = A + (1+\epsilon)I$	$\Phi(\lambda) = \bar{p}\left(\frac{1+\epsilon}{\bar{p}} + \mathbf{1} - \boldsymbol{\lambda}\right)$
GAT	Spatial	Trainable	$C^{(s)}_{v,u} = e_{v,u}/\sum_{k\in\tilde{\mathbb{N}}(v)} e_{v,k}$	NA
			$C^{(1)} = I$	$\Phi_1(\boldsymbol{\lambda}) = \mathbf{1}$
CayleyNet	Spectral	Trainable	$C^{(2r)} = \Re\left(\rho(hL)^r\right)$	$\Phi_{2r}(\boldsymbol{\lambda}) = \cos(r\theta(h\boldsymbol{\lambda}))$
			$C^{(2r+1)} = \Re\left(i\rho(hL)^r\right)$	$\Phi_{2r+1}(\boldsymbol{\lambda}) = -\sin(r\theta(h\boldsymbol{\lambda}))$
			$C^{(1)} = I$	$\Phi_1(\boldsymbol{\lambda}) = \mathbf{1}$
ChebNet	Spectral	Fixed	$C^{(2)} = 2L/\lambda_{\max} - I$	$\Phi_2(\boldsymbol{\lambda}) = 2\boldsymbol{\lambda}/\lambda\_\text{max} - \mathbf{1}$
			$C^{(s)} = 2C^{(2)}C^{(s-1)} - C^{(s-2)}$	$\Phi_s(\boldsymbol{\lambda}) = 2\Phi_2(\boldsymbol{\lambda})\Phi_{s-1}(\boldsymbol{\lambda}) - \Phi_{n-2}(\boldsymbol{\lambda})$

[8] Muhammet Balcilar et al. (2021). "Analyzing the Expressive Power of Graph Neural Networks in a Spectral Perspective". In: *Proc. ICLR*.

Architecture Searching via Sparse Coding

We put forward the following corollary to derive a unified mathematical expression between NAS-GNN and sparse coding.

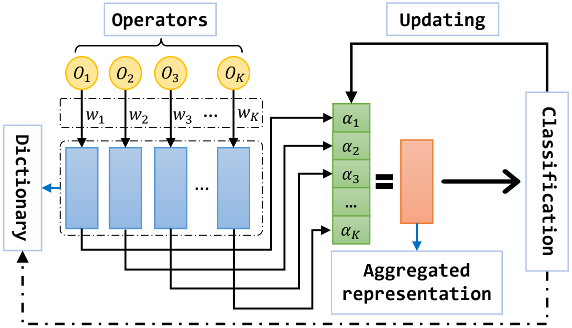
Corollary

The search space of GNNs can be unified as $\sum_{k=0}^{K} \mathrm{P}^k(\boldsymbol{L})\boldsymbol{X}\boldsymbol{W}^k = \boldsymbol{D}\boldsymbol{W}$ *when removing the activation function, where* $\boldsymbol{D} = \|\{\mathrm{P}^k(\boldsymbol{L})\boldsymbol{X}\}$ *is the fixed base, and* $\boldsymbol{W} = \|\{\boldsymbol{W}^k\}^T$ *is the trainable parameters. Here,* $\|$ *stands for concatenating a set of matrices horizontally.*

This corollary derives a unified dictionary for the search space in NAS-GNN, implying the natural connection between NAS-GNN and sparse coding.

Methodology

The optimality of untrained GNNs and the connection between NAS in GNNs and sparse coding allow us to waive the effort of updating weights and make us focus on architecture updates.
As shown in the Figure 5, we put forward Neural Architecture Coding(**NAC**), which directly learns architecture α with a fixed dictionary.



The framework of the proposed NAC in one layer.

Comparisons between NAC and state-of-the-art methods

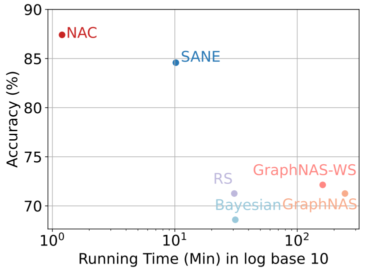
NAC attains superior performance than all baselines regarding both *accuracy* and *efficiency*.

Table: Experimental results on the compared methods: our NAC attains superior performance in both accuracy (%) and efficiency (in minutes).

	CiteSeer		Cora		PubMed		Computers	
	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time
RS	$70.12_{\pm 2.36}$	14.4	$71.26_{\pm 4.68}$	30.6	$86.75_{\pm 0.82}$	187.8	$77.84_{\pm 1.35}$	8.75
BO	$70.95_{\pm 1.62}$	18	$68.59_{\pm 6.66}$	31.2	$87.42_{\pm 0.68}$	189.6	$77.46_{\pm 2.02}$	17.65
GraphNAS	$68.69_{\pm 1.30}$	253.8	$71.26_{\pm 4.90}$	245.4	$86.07_{\pm 0.51}$	1363.8	$73.97_{\pm 1.79}$	86.37
GraphNAS-WS	$65.35_{\pm 5.13}$	80.4	$72.14_{\pm 2.59}$	161.4	$85.71_{\pm 1.05}$	965.4	$72.99_{\pm 3.44}$	42.47
SANE	$71.84_{\pm 1.33}$	4.2	$84.58_{\pm 0.53}$	10.2	$87.55_{\pm 0.78}$	107.4	$90.70_{\pm 0.89}$	0.72
NAC	$\mathbf{74.62}_{\pm 0.38}$	**1.2**	$\mathbf{87.41}_{\pm 0.92}$	**1.2**	$88.04_{\pm 1.06}$	**9.0**	$\mathbf{91.64}_{\pm 0.14}$	**0.23**
NAC-updating	$74.17_{\pm 1.18}$	4.2	$86.62_{\pm 1.14}$	3.6	$\mathbf{88.10}_{\pm 0.86}$	25.8	$90.89_{\pm 1.10}$	0.70

- In terms of model performance, our NAC beats all baselines and attains up to 2.83% improvement over the best baseline, i.e., SANE while attaining up to 18.8% improvement over the Bayesian method, the best HPO method.

Comparisons between NAC and state-of-the-art methods



Accuracy vs. running time on Cora. NAC (ours) outperforms the leading methods significantly in both accuracy and speed (in minutes).

- In terms of model efficiency, our NAC achieves superior performance, around 10× faster than SANE and up to 200× time faster than GraphNAS.

No-update Scheme at Work

To further validate our no-update scheme, we evaluate its effect on other weight-sharing methods.

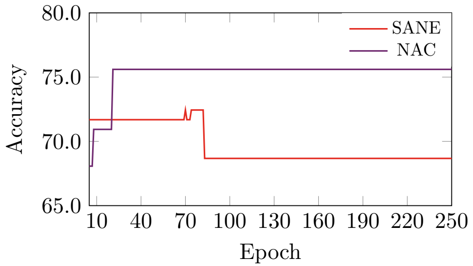
Table: Comparison between SANE and SANE* (w/o. weight updates).

	CiteSeer Acc(%)	Cora Acc(%)	Pubmed Acc(%)	Computers Acc(%)
SANE	$71.84_{\pm 1.33}$	$84.58_{\pm 0.53}$	$87.55_{\pm 0.78}$	$90.70_{\pm 0.89}$
SANE*	$\mathbf{71.95}_{\pm 1.32}$	$\mathbf{85.46}_{\pm 0.76}$	$\mathbf{88.12}_{\pm 0.35}$	$\mathbf{90.86}_{\pm 0.80}$

- Results in Table 3 show that SANE* outperforms the one with updates.
- This result implies that we can improve the performance of NAS-GNN methods by simply fixing the weights with orthogonal initialization.
- This yields a much lower computational cost in training.

Analysis of Convergence

A notable benefit of the NAC framework is its guaranteed convergence from the sparse coding perspective. Figure 7 offers a convergence comparison between NAC and SANE from the Pubmed dataset.



Convergence for SANE and NAC in terms of accuracy. NAC converges much faster than SANE in only around 20 epochs.

Overview

- We convert the traditional NAS problem into a sparse coding task with linear complexity of backward computing called Neural Architecture Coding(**NAC**).
- We have theoretically justified the power of untrained GNN in the NAS-GNN problem, where updating the neural weights is not necessary.
- NAC achieves higher accuracy (up to 18.8%) and much faster convergence speed(up to 200$\times$) without any updating on neural weights.

THANK YOU!