Disentangled Continual Graph Neural Architecture Search with Invariant Modular Supernet

Zeyang Zhang [1] **Xin Wang** [1] **Yijian Qin** [1] **Hong Chen** [1] **Ziwei Zhang** [1] **Xu Chu** [1] **Wenwu Zhu** [1]

Abstract

The existing graph neural architecture search (GNAS) methods assume that the graph tasks are static during the search process, ignoring the ubiquitous scenarios where sequential graph tasks come in a continual fashion. Moreover, existing GNAS works resort to entangled graph factors during the architecture search process, resulting in the catastrophic forgetting problems. In this paper, we study the problem of continual GNAS that is expected to continually search the architecture to learn new graph tasks without forgetting the past, which remains largely unexplored in the literature. However, this problem poses the challenge of *architecture conflicts*, *i.e.*, the optimal architecture for the new graph task may have performance deterioration and thus sub-optimal for past tasks. To address the challenge, we propose a novel Disentangled Continual Graph Neural Architecture Search with Invariant Modular Supernet (**GASIM**) method, which is able to continually search the optimal architectures without forgetting past knowledge. Specifically, we first design a modular graph architecture super-network incorporating multiple modules to enable searching architecture with factor expertise. Second, we propose a factor-based task-module router that discovers the latent graph factors and routes the incoming task to the best suitable architecture module to alleviate the forgetting problem induced by architecture conflicts. Finally, we propose an invariant architecture search mechanism to capture the shared knowledge among tasks. Extensive experiments demonstrate that our method achieves state-of-the-art performance against baselines in continual graph neural architecture search.

1. Introduction

Graph neural architecture search (GNAS), aiming at automating the discovery of optimal architectures for Graph neural networks (GNNs) when confronted with graph-structured data and specific tasks, has exhibited noteworthy advancements, demonstrating its efficacy in enhancing predictive capabilities and alleviating the need for extensive human involvement across a spectrum of graph applications (Zhang et al., 2021b). Most existing GNAS methods treat the entire graph as a holistic entity and search the architectures via optimizing for a fixed task applied to the entire graph, which is based on the underlying assumption that the graph tasks remain static throughout the search process.

However, graph tasks come continually in many real-world scenarios. For example, in citation networks, novel types of papers and their corresponding citations may continually surface, necessitating a continually updated classifier to distinguish documents belonging to newly emerging research fields (Zhang et al., 2022d; Zhou & Cao, 2021). In the domain of drug design, the encounter with new categories or properties of molecules is a recurrent phenomenon, and thus, the predictor should be consistently updated to assimilate knowledge regarding these new molecule categories or properties (Kirkpatrick et al., 2017). As the existing GNAS methods assume the graph tasks are fixed, they will inevitably fail in the ubiquitous settings where graph tasks are continuously generated.

In this paper, we study the problem of continual graph neural architecture search that is expected to continually search the graph architecture to learn new graph tasks without forgetting the past, which remains largely unexplored in the literature. However, we discover the problem is highly nontrivial with the critical challenge of *architecture conflicts*, *i.e.*, graph tasks may have different optimal architectures, while the optimal architecture for the new graph task may perform worse on past tasks, and thus it could be challenging to search the architecture to learn the new graph task without harming the performance on previous tasks. We further utilize a mainstream class of differentiable GNAS methods for an example, and demonstrate with theoretical analyses that they fail to revolve the problem of architecture conflicts due to their entangled search process, resulting the

[1] Department of Computer Science and Technology, BNRIST, Tsinghua University, Beijing, China. Correspondence to: Xin Wang <xin_wang@tsinghua.edu.cn>, Wenwu Zhu <wwzhu@tsinghua.edu.cn>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

catastrophic forgetting problems in continual graph architecture search process.

To address these challenges, we propose a Disentangled Continual Graph Neural Architecture Search with Invariant Modular (**GASIM**) Supernet method, which is able to continually search the optimal graph architectures without forgetting the past graph tasks. The key idea is to design a modular graph super-network, wherein each module undertakes the search for optimal architectures tailored to the graph tasks with similar factors so that the task-architecture relationship is disentangled to foster the sharing of mutual knowledge and mitigating conflicts throughout the continuous search process. Specifically, we first design a modular graph architecture super-network incorporating multiple modules to enable searching architecture with factor expertise and resolve architecture conflicts. Each module is designed to be activated by the incoming graph before searching. Second, we propose a factor-based task-module router that discovers the latent graph factors and routes the incoming task to the best suitable architecture module to alleviate the forgetting problem induced by architecture conflicts. We adopt a distribution matching method to predict the latent graph factors, and then maintain a task-module relation graph to route the new graph task to suitable modules. Finally, we propose an invariant architecture search mechanism to capture the shared knowledge among tasks. Based on the task-module graph, we extract the relevant graph factors, and augment the search process by designing an invariance loss to discover architectures harmonizing all graph tasks that have ever routed to this specific module. Extensive experiments on real-world datasets demonstrate that the proposed method achieves state-of-the-art performance against baselines in continual graph neural architecture search.

The contributions of this paper are summarized as follows:

- We study the largely unexplored problem of continual graph neural architecture search and propose a novel Disentangled Continual Graph Neural Architecture Search with Invariant Modular Supernet (**GASIM**) method which is able to discover the optimal graph neural architectures in a continual fashion.
- We discover the problem of *architecture conflicts* in continual graph architecture search, and demonstrate with theoretical analyses that a mainstream classic of differentiable search methods can not resolve the problem due to the entangled search process.
- We introduce three novel modules, i) modular graph architecture super-network, ii) factor-based task-module router and iii) invariant architecture search mechanism, which can continually search the optimal architectures to learn the new graph tasks while mitigating the problem of forgetting past knowledge.
- Extensive experiments on real-world datasets demonstrate that the proposed method achieves state-of-the-art performance against baselines in continual graph neural architecture search.

2. Problem Formulation

Graph Neural Architecture Search Let $\mathcal{G}$ denote the graph space and $\mathcal{Y}$ the label space. A graph neural network is represented as a function $a_{\alpha,w} : \mathcal{G} \to \mathcal{Y}$, with architecture parameters $\alpha \in \mathcal{A}$ and learnable weights $w \in \mathcal{W}$, where $\mathcal{A}$ is the architecture space and $\mathcal{W}$ is the weight space. Graph Neural Architecture Search (GNAS) automates the design of graph neural architectures by searching for the optimal α. As α typically represents the selection of GNN operations (e.g., GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2018), GIN (Xu et al., 2018)), it is referred to as operation choices for brevity. GNAS addresses the bi-level optimization problem (Elsken et al., 2019):

$$\alpha^* = \arg\min_{\alpha \in \mathcal{A}} \mathcal{L}(a_{\alpha,w^*(\alpha)}, \mathcal{G}, \mathcal{Y}), \tag{1}$$

$$\text{s.t. } w^*(\alpha) = \arg\min_{w \in \mathcal{W}(\alpha)} \mathcal{L}(a_{\alpha,w}, \mathcal{G}, \mathcal{Y}), \tag{2}$$

where $\mathcal{L}$ represent the loss of predictions by the architecture $a_{\alpha,w}(\cdot)$ on the graph, α^* and w^* is the best architecture and weight for the given task $\mathcal{T} = (\mathcal{G}, \mathcal{Y})$.

Continual Graph Neural Architecture Search In real-world scenarios, graphs are generated continuously. Consider a stream of graphs $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_t\}$, where each graph $\mathcal{G}_t = \{\mathcal{V}_t, \mathcal{E}_t\}$ has its own set of nodes and edges. Accompanying the graphs is a sequence of tasks $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_t\}$ and their corresponding labels $\mathcal{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_t\}$. For example, in node classification tasks, each node $v_t^i \in \mathcal{V}_t$ is expected to be classified to its node category $y_t^i \in \mathcal{Y}_t$, while the incoming tasks could be distinct, *e.g.*, $\mathcal{Y}_{t-1} \bigcup \mathcal{Y}_t = \emptyset$. An ideal continual graph neural architecture search is expected to continually modify the architecture to learn the new task without forgetting the past tasks. The objective can be formulated as

$$\alpha_t^*, w_t^* = \arg\min_{\alpha \in \mathcal{A}, w \in \mathcal{W}} \sum_{k=1}^{t} \mathcal{L}(a_{\alpha,w}, \mathcal{G}_k, \mathcal{Y}_k), \tag{3}$$

where α_t^*, w_t^* are the best architecture and weights at task $\mathcal{T}_t$, which can achieve the best average performance on all the tasks that have been seen so far. Since it is impractical to access all past graphs in the continual learning process, we consider the scenarios that only the current graph is available while the past graphs are not accessible in the stage of training or searching, following the continual learning literature (Wang et al., 2023a; Zhang et al., 2022c). The model should modify the architecture to learn the new task while maximally reducing the harm to the past tasks.

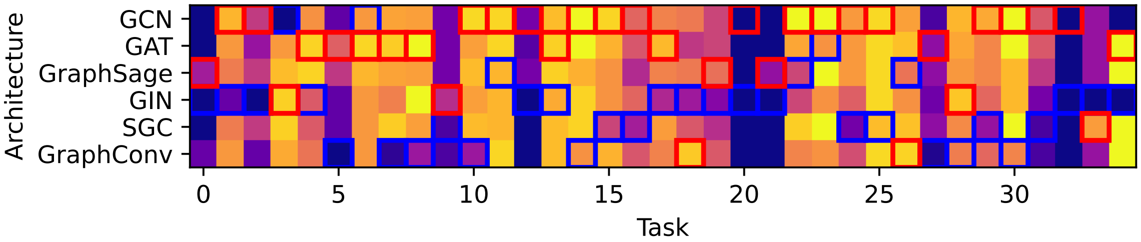

Figure 1: A showcase heatmap demonstrating that new graph tasks may have different optimal architectures on CoraFull dataset. Lighter colors denote higher performance of the architecture on the according task. Red and blue boxes denote the best performed and worst performed architectures on each task respectively. (Best viewed in color).

3. Preliminary Studies: Architecture Conflicts

In the settings of continual graph neural architecture search, the models may face the dilemma of *architecture conflicts*, where the optimal architecture of the new task may have poor performance for the past tasks. In such situations, directly searching architectures for the new task will bring catastrophic forgetting of the past tasks. To elaborate on this, we first conduct a preliminary experiment. We train and test six different GNN architectures for each task on CoraFull dataset, and report the performance for each task and architecture by a heatmap. From Figure 1, we have the following observations:

Obs. 1. Different tasks have their own optimal architectures, and no architecture is always the winner. This phenomenon indicates that the performance is destined to be suboptimal if the architecture is fixed during graph continual learning, and also demonstrates the necessity of continual graph neural architecture search.

Obs. 2. Some tasks have conflicting optimal architectures. For example, GIN performs the worst on the third task while the best on the fourth task, and in contrast, GCN performs the best on the third task while the worst on fourth task. This phenomenon brings great challenges for continual graph neural architecture search, since searching the optimal architecture for the new task may greatly deteriorate the performance of previous tasks if there exist architecture conflicts between these tasks.

We further provide theoretical analyses of graph neural architecture search in the situations of architecture conflicts as follows. Consider two operations $f_1(\cdot), f_2(\cdot)$ for two graph regression tasks, we adopt DARTS (Liu et al., 2018) as the search algorithm that searches the mixed operation $F(\mathcal{G}) = o_1 f_1(\mathcal{G}) + o_2 f_2(\mathcal{G})$ with gradient methods to minimize the MSE loss $\mathcal{L}_1(F)$ and $\mathcal{L}_2(F)$ for each task respectively, where $o_1 = \frac{\exp(\alpha_1)}{\exp(\alpha_1)+\exp(\alpha_2)}$ and $o_2 = \frac{\exp(\alpha_2)}{\exp(\alpha_1)+\exp(\alpha_2)}$ and α_1, α_2 are learnable parameters. Suppose the operations are distinguishable, *i.e.*, $\exists \mathcal{G}_k, f_1(\mathcal{G}_k) \neq f_2(\mathcal{G}_k)$. Suppose $y = f_1(\mathcal{G})$ and $y = f_2(\mathcal{G})$ are the ground-truth labeling function for each task respectively, which means that the operations are conflicting for two tasks. We have the following proposition with proof in Appendix.

Proposition 3.1. *For optimizing $\mathcal{L}_2(F)$, after one step of gradient descent w.r.t architecture parameters, the changes of operation weights o_1, o_2 will increase the loss $\mathcal{L}_1(F)$.*

This proposition shows that DARTS can not handle the architecture conflict problems, and it will inevitably fail due to forgetting the past when searching for the new task. The problem is that for both tasks, the super-network adopts the same mixed operation, *i.e.*, o_1, o_2, which is entangled together in the continual searching process.

To this end, our key idea is to design a modular graph super-network, wherein each module undertakes the search for optimal architectures tailored to graph tasks with similar factors so that the task-architecture relationship is disentangled to foster the sharing of mutual knowledge and mitigating conflicts throughout the continuous search process.

4. Methodology

In this section, we introduce Disentangled Continual Graph Neural Architecture Search with Invariant Modular Supernet (**GASIM**) to continually search graph architectures, by proposing three key components, modular graph architecture super-network, factor-based task-module router, and invariant architecture search mechanism. The overall framework is illustrated in Figure 2.

4.1. Modular Graph Architecture Super-network

To alleviate architecture conflicts and share mutual knowledge, we propose a modular graph architecture super-network to concurrently estimate and search for K different modules, where each time one module is updated to accommodate graph tasks with similar factors and the other modules are kept unchanged to reduce conflicts.

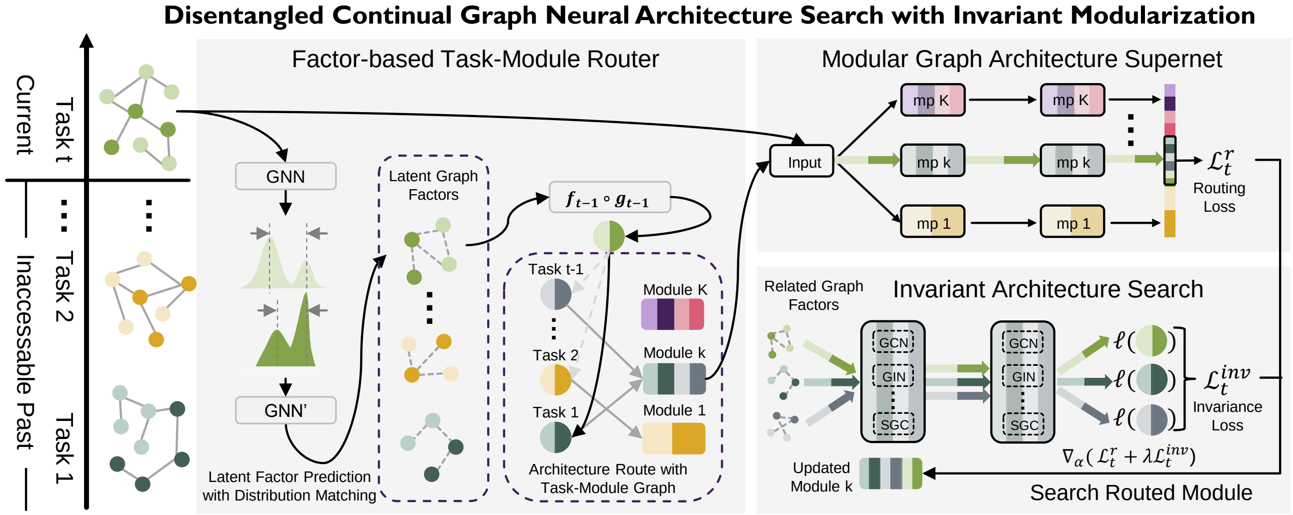
Disentangled Continual Graph Neural Architecture Search with Invariant Modularization
Current
Inaccessable Past
Task t
Task 2
Task 1
Factor-based Task-Module Router
GNN
GNN'
Latent Factor Prediction with Distribution Matching
Latent Graph Factors
$f_{t-1} \circ g_{t-1}$
Task t-1
Task 2
Task 1
Module K
Module k
Module 1
Architecture Route with Task-Module Graph
Modular Graph Architecture Supernet
Input
mp K
mp k
mp 1
$\mathcal{L}_t^r$
Routing Loss
Related Graph Factors
Invariant Architecture Search
GCN
GIN
SGC
$\ell($ $)$
$\mathcal{L}_t^{inv}$
Invariance Loss
$\nabla_\alpha(\mathcal{L}_t^r + \lambda\mathcal{L}_t^{inv})$
Updated Module k
Search Routed Module

Figure 2: The framework of Disentangled Continual Graph Neural Architecture Search with Invariant Modular Supernet (**GASIM**), including the following three key components: 1) Modular graph architecture super-network incorporating multiple modules to enable searching architectures with factor expertise, 2) Factor-based task-module router that discovers the latent graph factors and routes the incoming task to the best suitable architecture module to alleviate the forgetting problem induced by architecture conflicts, 3) Invariant architecture search mechanism to capture the shared knowledge among tasks in the routed module. (Best viewed in color)

Modular Super-network Layer Each super-network layer employs K mixed operations parameterized by different vectors α to learn K-chunk graph representations:

$$\mathbf{H}_k \leftarrow \overline{\text{GNN}}_{\alpha_k}(\mathbf{H}, \mathbf{A}), \tag{4}$$

where $\mathbf{A}$ denotes the adjacency matrix of the graph, $\mathbf{H}$ represents the input graph representations, and $\overline{\text{GNN}}_{\alpha_k}(\cdot)$ denotes the mixed GNN operations parameterized by α_k. To facilitate differentiable optimization, we adopt a continuous parameterization and weight-sharing mechanism (Liu et al., 2018) for the mixed operations:

$$\overline{\text{GNN}}_{\alpha_k}(\mathbf{H}, \mathbf{A}) = \sum_{i=1}^{|\mathcal{O}|} \alpha_{k,i}\text{GNN}_i(\mathbf{H}, \mathbf{A}), \tag{5}$$

where $|\mathcal{O}|$ is the number of GNN operation choices, $\alpha_{k,i} = \frac{\exp(\theta_{\alpha_{k,i}})}{\sum_j \exp(\theta_{\alpha_{k,j}})}$ denotes the probability of the i-th operation for the k-th architecture α_k, and θ represents learnable parameters. The overall super-network is constructed in the form of a directed acyclic graph (DAG) with an ordered sequence of modular super-network layers.

Flexible Modular Path An incoming graph can select a flexible path in the modular super-network, *i.e.*,

$$\pi(\mathcal{G}_t) = (a_{\alpha_i, w_i}(\cdot), \dots, a_{\alpha_j, w_j}(\cdot)), \tag{6}$$

where each α in the path is the chosen operations to be activated to calculate the representations of the graph. This design mitigates the conflicts of architectures by offering a broader array of flexible paths in the super-network (Guo et al., 2020). To illustrate, the 'mean' operation is adept at capturing structural properties, while the 'max' operation excels at capturing representative elements (Xu et al., 2018). When the consecutive tasks have operation conflicts, our method can simultaneously capture both aspects by selecting operations to learn the respective representations.

Route then Search Directly optimizing the modular path for each graph as Eq. (6) could be difficult for its large discrete solution space. We transform the probability of the path into the expectation of multiple paths under different modules by the Bayesian formula, *i.e.*,

$$p(\pi(\mathcal{G}_t) \mid \mathcal{G}_t) = \sum_{k=1}^{K} \underbrace{p(k \mid \mathcal{G}_t)}_{\text{Route}} \overbrace{p(\pi(\mathcal{G}_t) \mid \mathcal{G}_t, k)}^{\text{Search}}, \tag{7}$$

where the probability of the graph $\mathcal{G}_t$ to activate the k-th module $p(k \mid \mathcal{G}_t)$ can be modeled by the relationship between modules and tasks, the probability of module-specific architectures $p(\pi(\mathcal{G}_t) \mid \mathcal{G}_t, k)$ can be optimized by a specific super-network. To further alleviate the forgetting problem, we adopt a hard routing mechanism that only one module can be selected to be searched while others are kept unchanged, so that learning the new task will not affect the results of other past tasks using other modules.

4.2. Factor-based Task-Module Router

Graph factors are shown important in graph formation, representing various inherent graph properties (Fan et al., 2019; Ma et al., 2019). We propose to leverage the graph factors to maintain a task-module graph to implement the routing as Eq. (7). Specifically, we learn the latent factors with distribution matching following (Liu et al., 2023a), and measure the task similarity by the factors to conduct routing.

Latent Factor Prediction with Distribution Matching For task $\mathcal{T}_t$ with graph $\mathcal{G}_t = (\mathbf{A}_t, \mathbf{X}_t)$ and labels $\mathcal{Y}_t$, we learn the latent factors $\mathbf{F}_t$ as a sketch of the graph task. The factors should faithfully reflect the characteristics and distributions of the original graph, so that the module can be routed through the factor similarity. To this end, we adopt Maximum Mean Discrepancy (MMD) to learn the factors.

$$\mathbf{F}_t^* = \arg\min_{\mathbf{F}} \ell_{\text{MMD}}(\text{GNN}(\mathbf{X}_t, \mathbf{A}_t), \text{GNN}'(\mathbf{F}, \mathbf{A}_f)), \quad (8)$$

where $\mathbf{A}_f$ is the adjacency matrix of the relationship between the factors and is set to a self-loop matrix for simplicity. GNN$(\cdot)$ and GNN$'(\cdot)$ are shared random encoders, which encode the original graph and the learned graph factors into a shared latent space. From the perspective of self-supervised learning, the objective aims to reconstruct the distribution characteristics in the latent space (Hou et al., 2022; Xie et al., 2022). We random sample several node features as the initialization of the factors to ease the optimization process. We learn the latent factors for each class, and denote the classes of the factors as $\mathcal{Y}_t^f$.

Architecture Route with Task-Module Graph We maintain the task-module graph, and keep updating it to assign new tasks to modules. The task-module graph at task t is defined by a relation matrix $\mathbf{R}_t^m \in [0,1]^{t\times K}$, where the t-th row $\mathbf{r}_t^m \in [0,1]^K$ denotes the probability of $\mathcal{T}_t$ being assigning to each module. Denote the super-network at task t as $g_t(\cdot)$ and the classifier as task t as $f_t(\cdot)$. We calculate the similarity between the t-th task and previous tasks by leveraging the factors and the previous classifier,

$$\tilde{\mathbf{R}}_t^f = f_{t-1} \circ g_{t-1}(\mathbf{F}_t), \quad (9)$$

where $\tilde{\mathbf{R}}_t^f \in \mathbb{R}^{N\times C_{t-1}}$ refers to the probabilities of factors belonging to the classes, N is the number of factors, and C_{t-1} is the number of classes till $(t-1)$-th task. We calculate the factor-task relation matrix $\mathbf{R}_t^f \in \mathbb{R}^{N\times(t-1)}$ by adding up the probabilities of the tasks' corresponding classes, *i.e.*, $\mathbf{R}_t^f = \tilde{\mathbf{R}}_t^f \cdot \mathbf{R}_t^c$, where $\mathbf{R}_t^c \in \{0,1\}^{C_{t-1}\times(t-1)}$ is a matrix representing the relations between the classes and the tasks. Then we calculate the probability of the new task $\mathcal{T}_t$ being assigned to the modules as

$$\mathbf{r}_t^m = \text{Softmax}(\text{Mean}(\mathbf{R}_t^f \cdot \mathbf{R}_{t-1}^m)), \quad (10)$$

where Mean$(\cdot)$ takes the average along the first dimension, and $\mathbf{r}_t^m$ is the assigning vector for task $\mathcal{T}_t$. We adopt the module with maximum probability in $\mathbf{r}_t^m$ for searching, and add the probability vector into $\mathbf{R}_t^m$ for subsequent task routing. For the initialization, we assign one-hot vectors for the early tasks $\mathcal{T}_t, t < K$. By leveraging the relationship between factors, tasks and modules, we continually update the task-module graph for routing new tasks to modules with similar properties to avoid architecture conflicts while sharing mutual knowledge.

4.3. Invariant Architecture Search Mechanism

In the continual GNAS process, each module searches architectures for its routed new task, which may have distribution shifts from the past tasks that have routed to the module, even though they have similar factors. This shift between tasks may cause the module to overfit the new task while forgetting the past tasks. To this end, we propose an invariant architecture search mechanism to better share mutual knowledge via capturing the features invariant to tasks.

Routing Loss Suggest the router chooses k-th module for the task $\mathcal{T}_t$, and denote $g_t^k(\cdot)$ as the super-network sliced with k-th module, and $f_t^k(\cdot)$ as the classifier. Then loss is calculated as

$$\mathcal{L}_t^r = \ell(f_t^k \circ g_t^k(\mathcal{G}_t), \mathcal{Y}_t), \quad (11)$$

where $\ell(\cdot)$ is the cross-entropy loss function, and $\mathcal{G}_t$ and $\mathcal{Y}_t$ are the graph and labels at the current task. Note that only the routed module will be updated while other modules are kept unchanged to alleviate the forgetting problems.

Invariance Loss Based on the task-module graph $\mathbf{R}_t^m$, we select the factors that have the common module choice of the current task. Suggest the router chooses k-th module for the task $\mathcal{T}_t$, we obtain the factor indices $\mathcal{I}_t$ by

$$\mathcal{I}_t = \{j \mid \arg\max_i \mathbf{R}_t^m(j,i) = k\}. \quad (12)$$

Then we obtain a set of factors and their classes $\{\mathbf{F}_i, \mathcal{Y}_i^f \mid\in \mathcal{I}_t\}$, and calculate the invariance loss as

$$\mathcal{L}_t^{inv} = \text{Var}(\ell(f_t^k \circ g_t^k(\mathbf{F}_i), \mathcal{Y}_i^f) \mid i \in \mathcal{I}_t), \quad (13)$$

where Var$(\cdot)$ calculates the variance. This loss constrains the searched architecture from losing the capability of handling the previous tasks routed to the same module. The final loss for searching the architectures is

$$\mathcal{L}_t = \mathcal{L}_t^r + \lambda \mathcal{L}_t^{inv}, \quad (14)$$

where λ is a hyper-parameter to make a trade-off between the performance of the new task and past tasks. Similar to

(Liu et al., 2018), the architecture parameters at task $\mathcal{T}_t$ are updated differentiably with

$$\alpha = \alpha - \eta_\alpha \nabla_\alpha \mathcal{L}_t, w = w - \eta_w \nabla_w \mathcal{L}_t. \tag{15}$$

After searching the architectures, we fix the architectures and finetune the weights with factors for better classification. The overall algorithm is summarized in Algo. 1

Algorithm 1 The pipeline for **GASIM**

```
Require: The number of tasks $T$, hyperparameter $\lambda$, $K$.
Construct the modular super-network in Sec. 4.1.
for $l = 1, \dots, T$ do
    Predict the latent factors as Eq. (8)
    Route the task to the module as Eq. (10)
    Calculate routing loss as Eq. (11)
    Calculate invariance loss as Eq. (13)
    Calculate the final loss as Eq. (14)
    Search the architecture according to Eq. (15)
    Fix the architecture and finetune the weights
end for
```

Discussions about the framework and MoE The router of our framework aims to learn several groups of similar tasks with expert architecture modules. While the router and Mixture-of-Experts (MoE) (Shazeer et al., 2017; Fedus et al., 2022) have something in common that they both route the data to different parameterized networks for forwarding, they have different objectives. The objective of most MoE methods is to promote the performance by significantly enlarging the model capacity, while controlling the model sparsity to maintain the inference efficiency. In contrast, in our framework, the objective of the routing is to alleviate the architecture conflicts in the continual search process by routing dissimilar tasks to different modules, while routing similar tasks to the same module to boost shared knowledge. Therefore, differing from the MoE objective, the main focus of the routing is to facilitate modularization and continually search graph architectures to adapt to new tasks while not forgetting the previous knowledge.

5. Experiments

In this section, we conduct experiments on real-world continual graph datasets to verify the design of our method. We also include detailed ablation studies to analyze the effectiveness of each component.

5.1. Experimental Setups

Baselines We compare our method with 10 baselines from the following two different categories.

- **Manually designed GNNs.** We include six representative GNNs as our baselines, i.e., GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2018), GIN (Xu et al., 2018), GraphSage (Hamilton et al., 2017), GraphConv (Morris et al., 2019), SGC (Wu et al., 2019) which also constitute our search space.
- **Graph neural architecture search.** We include two representative GNAS baselines GraphNAS (Gao et al., 2020) and GASSO (Qin et al., 2021) where GASSO is specially designed for node-level classification tasks by searching the graph structures simultaneously. We also include two classical NAS baselines, random search and DARTS (Liu et al., 2018). Since they are not specially designed for graphs, we adopt our search space for them.

Since these methods are not designed for continual graph learning, we adopt the most recent and competitive continual graph learning method (Liu et al., 2023a;b) to train them to suit to the continual settings.

Super-network Construction The super-network generally consists of two parts, the operation pool and the directed acyclic graph (DAG) that wires the operations. For the operation pool, we include the six node aggregation operations mentioned in the manually designed GNN baselines. For brevity, we denote 'Agg' as node aggregation operations. Following (Qin et al., 2021), the DAG for each module is a straightforward path, i.e., $\mathbf{H}^{l+1} = \mathrm{Agg}^l(\mathbf{H}^l, \mathbf{A})$, and the embeddings of the last layer are utilized for downstream tasks, where $\mathbf{H}^l$ denotes the hidden embeddings output by the l-th layer, and $\mathbf{A}$ denotes the graph adjacency matrix. For searching the routed module, the embedding is fed to the module's classifier $f(\cdot)$, and the prediction is $f(\mathbf{H})$. For finetuning and testing the whole model, the embeddings of each module are concatenated together, *i.e.*, $\mathbf{H} = [\mathbf{H}_1||\mathbf{H}_2||\cdots||\mathbf{H}_K]$, which is subsequently fed to the whole model's classifier for prediction.

Datasets We adopt three node classification tasks in experiments, including CoraFull (McCallum et al., 2000), Arxiv (Hu et al., 2020), and Reddit (Hamilton et al., 2017). CoraFull and Arxiv are citation networks, and Reddit is a post-to-post graph. Table 1 shows the statistics of these datasets. All datasets are partitioned into a set of tasks, each focusing on the node classification problem, where each task involves nodes from two distinct classes within an incoming graph. For each task, 60% of the nodes are allocated for training, 20% for validation, and 20% for testing.

Evaluation We adopt the challenging setting that no task indicator is provided in the testing stage, also known as class-IL setting (Wang et al., 2023a), and the model continues to make classifications for all categories that have been seen so far. During the continual update phase, the model can only access the newly incoming graph and all the past graphs are

Table 1: A summary of dataset statistics.

Dataset	Nodes	Edges	Features	Classes	Tasks
CoraFull	19,793	130,622	8,710	70	35
Arxiv	169,343	1,166,243	128	40	20
Reddit	227,853	114,615,892	602	40	20

not available. Denote $m_{k,i}$ as the performance on i-th task after learning k-th task. We adopt three metrics to measure the model performance: 1) Average performance (AP), measuring the average model performance after learning from Task $\mathcal{T}_k$, *i.e.*, $\text{AP}_k = \frac{1}{k}\sum_{i=1}^{k} m_{k,i}$, 2) Mean of average performance ($\overline{\text{AP}}$), measuring the average performance of the continual learning process, *i.e.*, $\overline{\text{AP}} = \frac{1}{k}\sum_{i=1}^{k}\text{AP}_i$. 3) Average Forgetting (AF), measuring the influence of the training process for the current task on the performance of previous tasks, *i.e.*, $\text{AF}_k = \frac{1}{k-1}\sum_{i=1}^{k-1}(m_{k,i} - m_{i,i})$. A higher value indicates that training the current task has a more substantial impact on historical tasks. A negative or positive number signifies a detrimental or beneficial impact.

More experimental details are provided in the Appendix, including configurations and implementation details.

5.2. Main Results

From the results summarized in Table 2, we have the following observations:

- No hand-designed GNN is always the best for all datasets, which verifies the demand for automated graph architecture design in a continual fashion.
- The existing GNAS methods do not always beat the hand-designed GNNs, showing their limitations in handling continual graph data since their underlying assumption of static structural distribution and task may not hold in real continual graph data.
- The existing GNAS methods have lower average forgetting in most cases. In particular, DARTS achieves 61% average performance while having -16.6% AF on Arxiv dataset. This phenomenon indicates that DARTS can search a better architecture for each task, but the architecture may perform poorly for the previous tasks, which further verifies our discovered problem of architecture conflicts in the continual search process. Neglecting the problem will let the model overfit the newly-coming task while harming the past knowledge.
- Our method achieves significant improvements over the baselines. In particular, our method improves 8% in terms of average performance over the second-best baseline on CoraFull dataset. The results verify the effectiveness of our method that is able to continually search the architectures to learn the newly-coming tasks without forgetting past knowledge.

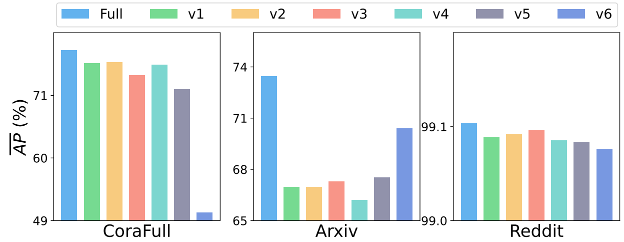

Figure 3: Comparisons of different ablated variants of **GASIM** on real-world datasets in terms of mean average performance. 'Full' denotes the full version of the method (Best viewed in color).

5.3. Additional Experiments

Ablation studies To verify the effectiveness of each designed component, we compare different ablated versions on each dataset: 1) v1 removes the invariance loss in Eq. (13), 2) v2 replaces the similar factor selection in Eq. (12) with random factor selection, 3) v3 replaces the router in Eq. (10) with a random router, which routes the task to a random module, 4) v4 replaces the router in Eq. (10) with a fixed router, which always routes the task to the first module, 5) v5 replaces the router in Eq. (10) with a sequential router, which always routes the task to the next module, *i.e.*, $t\%K$, 6) v6 replaces the modular super-network with a vanilla super-network. From Fig. 3, we have the following observations:

- Our proposed **GASIM** outperforms all the ablated variants on all datasets, demonstrating the effectiveness of each component of our proposed framework in continually searching graph architectures.
- The ablated versions v1 and v2 have a huge performance drop on Arxiv, showing the importance of leveraging the invariance to share mutual knowledge.
- The hand-designed routers v3, v4 and v5 have more suboptimal and unstable results across datasets, showing the effectiveness of our factor-based task-module router.
- The ablated version v6 fails drastically on CoraFull dataset, showing that the modular super-network is critical for resolving architecture conflicts to boost the continual searching process.

Visualized showcases We illustrate the performance matrices of three methods, GCN, DARTS and ours in Fig. 4. The performance matrix shows the test results for all past tasks after learning the new task. We find that our method

Table 2: The results of all the methods on the real-world datasets. Numbers after the ± signs represent standard deviations. The best results are in bold.

Dataset	CoraFull			Arxiv			Reddit		
Metric	AP	AF	$\overline{\text{AP}}$	AP	AF	$\overline{\text{AP}}$	AP	AF	$\overline{\text{AP}}$
GCN	61.94±1.45	-10.17±1.01	75.54±0.76	62.64±0.91	-13.78±0.47	72.16±0.78	96.65±0.13	-0.78±0.13	98.09±0.06
GIN	67.70±0.52	**-3.62±1.73**	76.73±0.79	59.68±3.90	-16.93±3.14	72.21±0.94	97.77±0.05	0.04±0.02	98.68±0.04
GAT	40.26±5.70	-4.57±4.12	50.72±1.42	61.87±0.51	-9.70±0.48	70.03±1.07	78.14±1.82	-5.90±4.06	88.54±2.08
GraphSage	61.29±0.97	-9.17±0.76	74.25±1.02	59.56±0.92	-13.44±1.32	69.30±0.56	91.56±0.48	-1.95±0.42	94.57±0.20
SGC	60.87±1.68	-10.93±1.07	75.13±1.18	56.33±1.42	-10.96±0.42	66.94±0.75	96.36±0.13	-0.67±0.03	97.87±0.05
GraphConv	67.15±1.12	-6.21±0.93	77.42±1.11	53.48±7.30	-19.45±5.00	69.40±1.70	97.61±0.06	-0.22±0.15	98.72±0.03
Random	63.57±1.06	-12.69±2.86	77.03±0.60	57.94±10.48	-19.76±11.33	71.74±0.97	97.89±0.08	**12.27±5.76**	87.55±2.70
GraphNAS	60.01±8.50	-18.85±5.71	63.81±11.05	52.94±4.12	**-9.58±1.28**	62.94±4.34	96.98±0.48	-1.01±0.18	98.34±0.26
DARTS	66.02±0.52	-6.89±0.90	77.40±0.38	61.45±2.09	-16.59±1.65	71.74±1.42	97.55±0.09	-0.25±0.11	98.75±0.02
GASSO	61.94±1.08	-10.07±0.61	75.11±0.93	61.21±0.84	-15.02±0.50	70.93±0.80	96.43±0.19	-0.94±0.11	97.98±0.08
Ours	**76.03±0.77**	-5.22±0.80	**78.95±0.71**	**65.03±1.40**	-12.19±1.64	**73.43±0.84**	**98.40±0.01**	-0.17±0.06	**99.10±0.02**

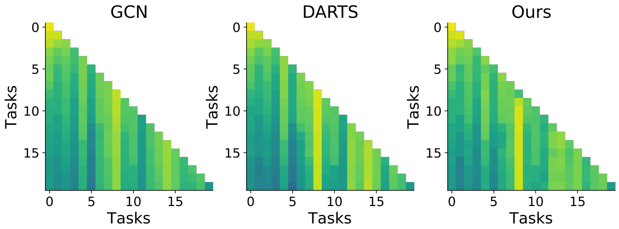

Figure 4: Performance matrices of different methods on Arxiv dataset. Lighter colors denote higher performance. The value $P_{i,j}, j \leq i$ denotes the performance on j-th task after learning the i-th task. (Best viewed in color).

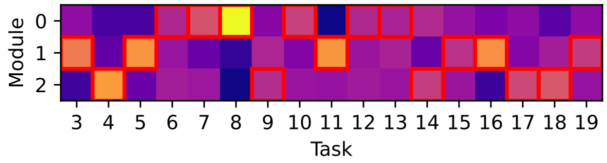

Figure 5: A showcase heatmap demonstrating that the router continually selects different modules on Arxiv dataset. Lighter colors denote higher probability of the module on the according task. Red boxes denotes the module choice on each task respectively. (Best viewed in color).

can continually assimilate new knowledge by searching the architectures for the new task. For example, the performance of the 8-th task even grows after learning the 10-th task. To make deeper analyses, we illustrate the router choices in Fig. 5. For the 8-th and the 10-th tasks, the router both chooses the 0-th module, which shows that the model may share the knowledge in the same module to boost the performance of both tasks in the continual searching process.

6. Related works

Graph Neural Architecture Search Neural architecture search has gained prominence as it endeavors to automatically discover optimal architectures for specific tasks. This trend has been particularly notable in computer vision (Ren et al., 2021; Elsken et al., 2019), natural language processing (Chitty-Venkata et al., 2022). Graph neural architecture search (GNAS) methods (Guan et al., 2021; Qin et al., 2022b), confronting the unique challenge of tackling the relationship of architectures and complex structures on graphs, can be broadly categorized into reinforcement-learning-based methods (Gao et al., 2020; Zhou et al., 2019; Cai et al., 2022; Gao et al., 2022; 2023), evolutionary-based methods (Nunes & Pappa, 2020; Li & King, 2020; Shi et al., 2022; Zhang et al., 2022a;b), and differentiable methods (Ding et al., 2021; Huan et al., 2021; Li et al., 2021c; Cai et al., 2021; Qin et al., 2022a; Wei et al., 2021; Zhang et al., 2023f;e; Li et al., 2024c; Zheng et al., 2023). However, a prevailing limitation of existing GNAS methods is their dependency on the underlying assumption of static distribution and tasks, and thus limiting their application in real-world scenarios. In this paper, we study the problem of continual graph neural architecture search in the setting where graphs are continuously generated.

Continual Graph Learning Continual graph learning aims to handle streaming graph data other than static graphs. Distinct from continual learning in computer vision (Kirkpatrick et al., 2017; Aljundi et al., 2018; Lopez-Paz & Ranzato, 2017; Li & Hoiem, 2017; Wang et al., 2022), continual graph learning methods should take into consideration the changes of structures as well as the tasks applied to these structures. The existing methods can be roughly classified into two categories, regularisation-based methods (Liu et al., 2021) and replay-based methods (Zhou & Cao, 2021; Zhang et al., 2022e; Liu et al., 2023a). (Liu et al., 2021) employs regularisation to preserve topological information from historical graphs. (Zhou & Cao, 2021) incorporates memory-replay mechanisms by storing representative nodes, while (Zhang et al., 2022e) employs memory banks to store sparsified subgraphs, thereby preserving structural information. (Liu et al., 2023a) condenses the graphs into memory with consideration of class imbalance, which are then adopted for replaying in the learning process. To assess and benchmark CGL methods, two recent benchmarks (Zhang et al., 2022c; Ko et al., 2022) have been developed, contributing to the evolving landscape of graph continual learning research. However, these methods adopt a fixed architecture with the assumption that the optimal architecture is static in the continual learning process. Recently, there has been increasing interest in utilizing the in-context learning capabilities of large language models for graph-related tasks (Jin et al., 2023; Zhang et al., 2023a; 2024; Yao et al., 2024; Li et al., 2024b; Chen et al., 2024b). However, it remains underexplored whether these methods can effectively adapt to continuously emerging graph domains and tasks. In this paper, we continually modify the architecture to suit for the newly-coming graph tasks in the continual learning process.

Disentangled Graph Learning Disentangled representation learning aims to delineate and elucidate the distinct latent factors that influence observable data, representing each factor as a unique vector representation (Bengio et al., 2013; Wang et al., 2023b), and the obtained factors could be helpful for explaining the inherent mechanism driving the data formation as well as producing robust representations for downstream tasks. This methodology has proven valuable across diverse domains, spanning computer vision (Hsieh et al., 2018; Ma et al., 2018; Chen et al., 2016; Wang et al., 2023c; Chen et al., 2023; 2024a), and graph representation learning (Ma et al., 2019; Liu et al., 2020; Yang et al., 2020; Chen et al., 2021; Li et al., 2021a;b; 2022c). In the context most relevant to our work, (Qin et al., 2022a) characterize latent factors within graph data by employing a self-supervised disentangled graph encoder, and then conducts graph neural architecture search for each graph to address distribution shifts, while the graphs and tasks are still fixed in the optimization process. In contrast, our focus in this paper centers on the automation of continual graph neural architecture designing via disentangling the relationship between graphs and architectures with the invariant modular super-network and routing mechanism.

Out-of-Distribution Generalization Many current machine learning techniques are based on the premise that training and testing data are independent and identically distributed. However, this assumption may not hold in complex real-world settings (Shen et al., 2021; Yao et al., 2022; Xu et al., 2022; Wen et al., 2024), particularly during the continual learning process where data distributions can progressively shift across sequential tasks. One classic of Out-of-Distribution (OOD) generalization methods is invariant learning that aims to obtain invariant features are robust across distributions (Arjovsky et al., 2019; Sagawa et al., 2019; Krueger et al., 2021). Recently, there is a growing interest in achieving OOD generalization on graphs (Li et al., 2024a; 2022a;b;d; 2023a;b; Wu et al., 2022a;b; Chen et al., 2022; Zhang et al., 2021a; 2022g; Fan et al., 2021) as well as dynamic graphs that come sequentially (Zhang et al., 2023b;c;d; 2022f). In our work, we leverage insights from the invariant learning literature and propose an invariance loss with regard to graph architectures and tasks to enhance knowledge sharing across different tasks and modules, thereby mitigating the distribution shifts during the continual learning process.

7. Conclusion

Existing graph neural architecture search (GNAS) methodologies fall short in addressing the dynamic nature of continually evolving graph data. In this paper, we propose Disentangled Continual Graph Neural Architecture Search with Invariant Modular Supernet (**GASIM**) method, specifically designed to overcome the challenges of continual GNAS. By proposing a modular graph architecture super-network and a factor-based task-module router, our approach efficiently navigates the entangled nature of graph factors and resolves conflicts arising from evolving relationships between the entire graph and optimal architectures. The proposed invariant architecture search method further ensures the retention of shared knowledge across tasks, enabling continual learning without forgetting past knowledge. Extensive experiments demonstrate that our method achieves state-of-the-art performance in continual graph neural architecture search. One limitation of this paper is that we mainly focus on homogeneous graphs, and we leave extending our method to heterogeneous graphs for further explorations.

Acknowledgements

This work is supported by the National Key Research and Development Program of China No.2023YFF1205001, National Natural Science Foundation of China (No. 62222209, 62250008, 62102222, 62206149), Beijing National Research Center for Information Science and Technology under Grant No. BNR2023RC01003, BNR2023TD03006, and Beijing Key Lab of Networked Multimedia.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 139–154, 2018.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

Cai, J., Wang, X., Guan, C., Tang, Y., Xu, J., Zhong, B., and Zhu, W. Multimodal continual graph learning with neural architecture search. In *Proceedings of the ACM Web Conference 2022*, pp. 1292–1300, 2022.

Cai, S., Li, L., Deng, J., Zhang, B., Zha, Z.-J., Su, L., and Huang, Q. Rethinking graph neural architecture search from message-passing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6657–6666, 2021.

Chen, H., Chen, Y., Wang, X., Xie, R., Wang, R., Xia, F., and Zhu, W. Curriculum disentangled recommendation with noisy multi-feedback. *Advances in Neural Information Processing Systems*, 34:26924–26936, 2021.

Chen, H., Zhang, Y., Wu, S., Wang, X., Duan, X., Zhou, Y., and Zhu, W. Disenbooth: Identity-preserving disentangled tuning for subject-driven text-to-image generation. In *The Twelfth International Conference on Learning Representations*, 2023.

Chen, H., Wang, X., Zhang, Y., Zhou, Y., Zhang, Z., Tang, S., and Zhu, W. Disenstudio: Customized multi-subject text-to-video generation with disentangled spatial control, 2024a.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016.

Chen, Y., Zhang, Y., Yang, H., Ma, K., Xie, B., Liu, T., Han, B., and Cheng, J. Invariance principle meets out-of-distribution generalization on graphs. *arXiv preprint*, 2022.

Chen, Z., Mao, H., Li, H., Jin, W., Wen, H., Wei, X., Wang, S., Yin, D., Fan, W., Liu, H., et al. Exploring the potential of large language models (llms) in learning on graphs. *ACM SIGKDD Explorations Newsletter*, 25(2): 42–61, 2024b.

Chitty-Venkata, K. T., Emani, M., Vishwanath, V., and Somani, A. K. Neural architecture search for transformers: A survey. *IEEE Access*, 10:108374–108412, 2022.

Ding, Y., Yao, Q., Zhao, H., and Zhang, T. Diffmg: Differentiable meta graph search for heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 279–288, 2021.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Fan, S., Wang, X., Shi, C., Cui, P., and Wang, B. Generalizing graph neural networks on out-of-distribution graphs. *arXiv preprint arXiv:2111.10657*, 2021.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. Graph neural networks for social recommendation. In *The world wide web conference*, pp. 417–426, 2019.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

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

Gao, Y., Zhang, P., Yang, H., Zhou, C., Hu, Y., Tian, Z., Li, Z., and Zhou, J. Graphnas++: Distributed architecture search for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):6973–6987, 2022.

Gao, Y., Zhang, P., Zhou, C., Yang, H., Li, Z., Hu, Y., and Philip, S. Y. Hgnas++: efficient architecture search for heterogeneous graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

Guan, C., Zhang, Z., Li, H., Chang, H., Zhang, Z., Qin, Y., Jiang, J., Wang, X., and Zhu, W. Autogl: A library for automated graph learning. In *ICLR 2021 Workshop GTRL*, 2021.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. In *European conference on computer vision*, pp. 544–560. Springer, 2020.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Hou, Z., Liu, X., Dong, Y., Wang, C., Tang, J., et al. Graphmae: Self-supervised masked graph autoencoders. *arXiv preprint arXiv:2205.10803*, 2022.

Hsieh, J.-T., Liu, B., Huang, D.-A., Fei-Fei, L. F., and Niebles, J. C. Learning to decompose and disentangle representations for video prediction. *Advances in neural information processing systems*, 31, 2018.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Huan, Z., Quanming, Y., and Weiwei, T. Search to aggregate neighborhood for graph neural network. In *2021 IEEE 37th International Conference on Data Engineering*, pp. 552–563, 2021.

Jin, B., Liu, G., Han, C., Jiang, M., Ji, H., and Han, J. Large language models on graphs: A comprehensive survey. *arXiv preprint arXiv:2312.02783*, 2023.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Ko, J., Kang, S., Kwon, T., Moon, H., and Shin, K. Begin: Extensive benchmark scenarios and an easy-to-use framework for graph continual learning. *arXiv preprint arXiv:2211.14568*, 2022.

Krueger, D., Caballero, E., Jacobsen, J.-H., Zhang, A., Binas, J., Zhang, D., Le Priol, R., and Courville, A. Out-of-distribution generalization via risk extrapolation (rex). In *International Conference on Machine Learning*, pp. 5815–5826. PMLR, 2021.

Li, H., Wang, X., Zhang, Z., Ma, J., Cui, P., and Zhu, W. Intention-aware sequential recommendation with structured intent transition. *IEEE Transactions on Knowledge and Data Engineering*, 34(11):5403–5414, 2021a.

Li, H., Wang, X., Zhang, Z., Yuan, Z., Li, H., and Zhu, W. Disentangled contrastive learning on graphs. *Advances in Neural Information Processing Systems*, 34:21872–21884, 2021b.

Li, H., Wang, X., Zhang, Z., and Zhu, W. Ood-gnn: Out-of-distribution generalized graph neural network. *IEEE Transactions on Knowledge and Data Engineering*, 2022a.

Li, H., Wang, X., Zhang, Z., and Zhu, W. Out-of-distribution generalization on graphs: A survey. *arXiv preprint arXiv:2202.07987*, 2022b.

Li, H., Zhang, Z., Wang, X., and Zhu, W. Disentangled graph contrastive learning with independence promotion. *IEEE Transactions on Knowledge and Data Engineering*, 2022c.

Li, H., Zhang, Z., Wang, X., and Zhu, W. Learning invariant graph representations for out-of-distribution generalization. In *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022d.

Li, H., Zhang, Z., Wang, X., and Zhu, W. Invariant node representation learning under distribution shifts with multiple latent environments. *ACM Transactions on Information Systems*, 42(1):1–30, 2023a.

Li, H., Wang, X., Zhang, Z., Chen, H., Zhang, Z., and Zhu, W. Disentangled graph self-supervised learning for out-of-distribution generalization. In *International conference on machine learning*. PMLR, 2024a.

Li, P., Meng, Y., Wang, X., Shen, F., Li, Y., Wang, J., and Zhu, W. Causal discovery in temporal domain from interventional data. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 4074–4078, 2023b.

Li, P., Wang, X., Zhang, Z., Meng, Y., Shen, F., Li, Y., Wang, J., Li, Y., and Zhu, W. Llm-enhanced causal discovery in temporal domain from interventional data. *arXiv preprint arXiv:2404.14786*, 2024b.

Li, P., Wang, X., Zhang, Z., Qin, Y., Zhang, Z., Wang, J., Li, Y., and Zhu, W. Causal-aware graph neural architecture search under distribution shifts, 2024c.

Li, Y. and King, I. Autograph: Automated graph neural network. In *International Conference on Neural Information Processing*, pp. 189–201, 2020.

Li, Y., Wen, Z., Wang, Y., and Xu, C. One-shot graph neural architecture search with dynamic search space. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8510–8517, 2021c.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

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

Liu, H., Yang, Y., and Wang, X. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 8653–8661, 2021.

Liu, Y., Wang, X., Wu, S., and Xiao, Z. Independence promoted graph disentangled networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 4916–4923, 2020.

Liu, Y., Qiu, R., and Huang, Z. Cat: Balanced continual graph learning with graph condensation. *arXiv preprint arXiv:2309.09455*, 2023a.

Liu, Y., Qiu, R., Tang, Y., Yin, H., and Huang, Z. Puma: Efficient continual graph learning with graph condensation. *arXiv preprint arXiv:2312.14439*, 2023b.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Ma, J., Cui, P., Kuang, K., Wang, X., and Zhu, W. Disentangled graph convolutional networks. In *International conference on machine learning*, pp. 4212–4221. PMLR, 2019.

Ma, L., Sun, Q., Georgoulis, S., Van Gool, L., Schiele, B., and Fritz, M. Disentangled person image generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 99–108, 2018.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 4602–4609, 2019.

Nunes, M. and Pappa, G. L. Neural architecture search in graph neural networks. In *Brazilian Conference on Intelligent Systems*, pp. 302–317, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Qin, Y., Wang, X., Zhang, Z., and Zhu, W. Graph differentiable architecture search with structure learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Qin, Y., Wang, X., Zhang, Z., Xie, P., and Zhu, W. Graph neural architecture search under distribution shifts. In *International Conference on Machine Learning*, pp. 18083–18095. PMLR, 2022a.

Qin, Y., Zhang, Z., Wang, X., Zhang, Z., and Zhu, W. Nas-bench-graph: Benchmarking graph neural architecture search. *arXiv preprint arXiv:2206.09166*, 2022b.

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., and Wang, X. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.

Sagawa, S., Koh, P. W., Hashimoto, T. B., and Liang, P. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*, 2019.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Shen, Z., Liu, J., He, Y., Zhang, X., Xu, R., Yu, H., and Cui, P. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*, 2021.

Shi, M., Wilson, D. A., Zhu, X., Huang, Y., Zhuang, Y., Liu, J., and Tang, Y. Genetic-gnn: Evolutionary architecture search for graph neural networks. *Knowledge-Based Systems*, pp. 108752, 2022.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Wang, L., Zhang, X., Su, H., and Zhu, J. A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*, 2023a.

Wang, X., Chen, H., Tang, S., Wu, Z., and Zhu, W. Disentangled representation learning, 2023b.

Wang, X., Wu, Z., Chen, H., Lan, X., and Zhu, W. Mixup-augmented temporally debiased video grounding with content-location disentanglement. 2023c.

Wang, Z., Xu, Q., Yang, Z., He, Y., Cao, X., and Huang, Q. Openauc: Towards auc-oriented open-set recognition. *Advances in Neural Information Processing Systems*, 35: 25033–25045, 2022.

Wei, L., Zhao, H., Yao, Q., and He, Z. Pooling architecture search for graph classification. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2091–2100, 2021.

Wen, P., Xu, Q., Yang, Z., He, Y., and Huang, Q. Algorithm-dependent generalization of auprc optimization: Theory and algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Wu, Q., Zhang, H., Yan, J., and Wipf, D. Handling distribution shifts on graphs: An invariance perspective. *arXiv preprint arXiv:2202.02466*, 2022a.

Wu, Y.-X., Wang, X., Zhang, A., He, X., and Chua, T.-S. Discovering invariant rationales for graph neural networks. *arXiv preprint arXiv:2201.12872*, 2022b.

Xie, Y., Xu, Z., and Ji, S. Self-supervised representation learning via latent graph prediction. *arXiv preprint arXiv:2202.08333*, 2022.

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

Xu, Q., Yang, Z., Jiang, Y., Cao, X., Yao, Y., and Huang, Q. Not all samples are trustworthy: Towards deep robust svp prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(6):3154–3169, 2022.

Yang, Y., Feng, Z., Song, M., and Wang, X. Factorizable graph convolutional networks. *Advances in Neural Information Processing Systems*, 33:20286–20296, 2020.

Yao, H., Wang, Y., Li, S., Zhang, L., Liang, W., Zou, J., and Finn, C. Improving out-of-distribution robustness via selective augmentation. In *Proceeding of the Thirty-ninth International Conference on Machine Learning*, 2022.

Yao, Y., Wang, X., Zhang, Z., Qin, Y., Zhang, Z., Chu, X., Yang, Y., Zhu, W., and Mei, H. Exploring the potential of large language models in graph generation. *arXiv preprint arXiv:2403.14358*, 2024.

Zhang, W., Lin, Z., Shen, Y., Li, Y., Yang, Z., and Cui, B. Deep and flexible graph neural architecture search. In *International Conference on Machine Learning*, pp. 26362–26374. PMLR, 2022a.

Zhang, W., Shen, Y., Lin, Z., Li, Y., Li, X., Ouyang, W., Tao, Y., Yang, Z., and Cui, B. Pasca: A graph neural architecture search system under the scalable paradigm. In *Proceedings of the ACM Web Conference 2022*, pp. 1817–1828, 2022b.

Zhang, X., Song, D., and Tao, D. Cglb: Benchmark tasks for continual graph learning. *Advances in Neural Information Processing Systems*, 35:13006–13021, 2022c.

Zhang, X., Song, D., and Tao, D. Hierarchical prototype networks for continual graph representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4622–4636, 2022d.

Zhang, X., Song, D., and Tao, D. Sparsified subgraph memory for continual graph representation learning. In *2022 IEEE International Conference on Data Mining (ICDM)*, pp. 1335–1340. IEEE, 2022e.

Zhang, Z., Wang, X., Zhang, Z., Cui, P., and Zhu, W. Revisiting transformation invariant geometric deep learning: Are initial representations all you need? *arXiv preprint arXiv:2112.12345*, 2021a.

Zhang, Z., Wang, X., and Zhu, W. Automated machine learning on graphs: A survey. *arXiv preprint arXiv:2103.00742*, 2021b.

Zhang, Z., Wang, X., Zhang, Z., Li, H., Qin, Z., and Zhu, W. Dynamic graph neural networks under spatio-temporal distribution shift. In *Advances in Neural Information Processing Systems*, 2022f.

Zhang, Z., Zhang, Z., Wang, X., and Zhu, W. Learning to solve travelling salesman problem with hardness-adaptive curriculum. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2022g.

Zhang, Z., Li, H., Zhang, Z., Qin, Y., Wang, X., and Zhu, W. Graph meets llms: Towards large graph models. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023a.

Zhang, Z., Li, X., Teng, F., Lin, N., Zhu, X., Wang, X., and Zhu, W. Out-of-distribution generalized dynamic graph neural network for human albumin prediction. In *IEEE International Conference on Medical Artificial Intelligence*, 2023b.

Zhang, Z., Wang, X., Zhang, Z., Li, H., and Zhu, W. Out-of-distribution generalized dynamic graph neural network with disentangled intervention and invariance promotion. *arXiv preprint arXiv:2311.14255*, 2023c.

Zhang, Z., Wang, X., Zhang, Z., Qin, Z., Wen, W., Xue, H., Li, H., and Zhu, W. Spectral invariant learning for dynamic graphs under distribution shifts. In *Advances in Neural Information Processing Systems*, 2023d.

Zhang, Z., Wang, X., Zhang, Z., Shen, G., Shen, S., and Zhu, W. Unsupervised graph neural architecture search with disentangled self-supervision. In *Advances in Neural Information Processing Systems*, 2023e.

Zhang, Z., Zhang, Z., Wang, X., Qin, Y., Qin, Z., and Zhu, W. Dynamic heterogeneous graph attention neural architecture search. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*, 2023f.

Zhang, Z., Wang, X., Zhang, Z., Li, H., Qin, Y., and Zhu, W. Llm4dyg: Can large language models solve spatial-temporal problems on dynamic graphs? In *Conference on Knowledge Discovery & Data Mining (ACM SIGKDD)*, 2024.

Zheng, X., Zhang, M., Chen, C., Zhang, Q., Zhou, C., and Pan, S. Auto-heg: Automated graph neural network on heterophilic graphs. In *Proceedings of the ACM Web Conference 2023*, pp. 611–620, 2023.

Zhou, F. and Cao, C. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 4714–4722, 2021.

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

A. Appendix

A.1. Proof of Proposition 1

To prove the proposition, we have to first give a Lemma A.1 inspired by (Qin et al., 2021).

Lemma A.1. *The operation weights are calculated by a softmax function. E.g.* $o_1 = \frac{exp\{\alpha_1\}}{exp\{\alpha_1\}+exp\{\alpha_2\}}$*. Let* g_1 *and* g_2 *are the gradients w.r.t.* α_1 *and* α_2*. The operation weights after a gradient descent step are* o_1' *and* o_2'*, e.g.,* $o_1' = \frac{exp\{\alpha_1-g_1\}}{exp\{\alpha_1-g_1\}+exp\{\alpha_2-g_2\}}$*. If* $g_1 < g2$*, then* $o_1' > o_1$ *and* $o_2' < o_2$*, and vice versa.*

Proof. If $g_1 < g_2$, we should improve $o_1' > o_1$, that is

$$\begin{aligned} \frac{exp\{\alpha_1 - g_1\}}{exp\{\alpha_1 - g_1\} + exp\{\alpha_2 - g_2\}} &> \frac{exp\{\alpha_1\}}{exp\{\alpha_1\} + exp\{\alpha_2\}} \\ exp\{\alpha_1 - g_1\}(exp\{\alpha_1\} + exp\{\alpha_2\}) &> exp\{\alpha_1\}(exp\{\alpha_1 - g_1\} + exp\{\alpha_2 - g_2\}) \\ exp\{2\alpha_1 - g_1\} + exp\{\alpha_1 + \alpha_2 - g_1\} &> exp\{2\alpha_1 - g_1\} + exp\{\alpha_1 + \alpha_2 - g_2\} \\ exp\{\alpha_1 + \alpha_2 - g_1\} &> exp\{\alpha_1 + \alpha_2 - g_2\} \\ \alpha_1 + \alpha_2 - g_1 &> \alpha_1 + \alpha_2 - g_2 \\ g_1 &< g_2 \end{aligned} \tag{16}$$

We already have $g_1 < g_2$, thus $o_1' > o_1$. Since $o_1 + o_2 = o_1' + o_2' = 1$, we have $o_2' < o_2$. □

Proposition A.2. *For optimizing* $\mathcal{L}_2(F)$*, after one step of gradient descent w.r.t architecture parameters, the changes of operation weights* o_1, o_2 *will increase the loss* $\mathcal{L}_1(F)$*.*

Proof. The MSE loss of $\mathcal{L}_2(F)$ is calculated as

$$\mathcal{L}_2(F) = \sum_k (F(\mathcal{G}_k) - y_k)^2. \tag{17}$$

Then we have

$$\begin{aligned} \frac{\partial \mathcal{L}_2(F)}{\partial o_i} &= 2\sum_k \Big(F(\mathcal{G}_k) - y_k\Big)\frac{\partial F(\mathcal{G}_k)}{\partial o_i} \\ &= 2\sum_k \Big(o_1 f_1(\mathcal{G}_k) + o_2 f_2(\mathcal{G}_k) - y_k\Big) f_i(\mathcal{G}_k) \\ &= 2\sum_k \Big(o_1 f_1(\mathcal{G}_k) + (1 - o_1) f_2(\mathcal{G}_k) - y_k\Big) f_i(\mathcal{G}_k) \\ &= 2\sum_k \Big(o_1 f_1(\mathcal{G}_k) + (1 - o_1) f_2(\mathcal{G}_k) - f_2(\mathcal{G}_k)\Big) f_i(\mathcal{G}_k) \\ &= 2o_1 \sum_k \Big(f_1(\mathcal{G}_k) - f_2(\mathcal{G}_k)\Big) f_i(\mathcal{G}_k). \end{aligned} \tag{18}$$

The second equation expand $F(x)$ by $F(x) = o_1 f(x) + o_2 f(x)$. The third equation utilizes relationship of o_1 and o_2, since $o_1 = \frac{exp\{\alpha_1\}}{exp\{\alpha_1\}+exp\{\alpha_2\}}$, $o_2 = \frac{exp\{\alpha_2\}}{exp\{\alpha_1\}+exp\{\alpha_2\}}$ and thus $o_1 + o_2 = 1$. The fourth equation utilizes the supposed ground-truth labeling function for the 2-rd task, *i.e.*, $y_k = f_2(\mathcal{G}_k)$. Then we have

$$\begin{aligned}\frac{\partial \mathcal{L}_2(F)}{\partial o_1} - \frac{\partial \mathcal{L}_2(F)}{\partial o_2} &= 2o_1 \sum_k \Big(f_1(\mathcal{G}_k) - f_2(\mathcal{G}_k)\Big)\Big(f_1(\mathcal{G}_k) - f_2(\mathcal{G}_k)\Big) \\ &= 2o_1 \sum_k \Big(f_1(\mathcal{G}_k) - f_2(\mathcal{G}_k)\Big)^2 \\ &> 0.\end{aligned} \tag{19}$$

The inequality utilizes that $o_1 = \frac{exp\{\alpha_1\}}{exp\{\alpha_1\}+exp\{\alpha_2\}} > 0$, and the assumption that $\exists \mathcal{G}_k, f_1(\mathcal{G}_k) \neq f_2(\mathcal{G}_k)$ so that $\sum_k \Big(f_1(\mathcal{G}_k) - f_2(\mathcal{G}_k)\Big)^2 > 0$. When $\frac{\partial \mathcal{L}_2(F)}{\partial o_1} - \frac{\partial \mathcal{L}_2(F)}{\partial o_2} = 0$, then $f_1(\mathcal{G}_k) = f_2(\mathcal{G}_k) \forall \mathcal{G}_k$ and $\frac{\partial \mathcal{L}_2(F)}{\partial o_1} = \frac{\partial \mathcal{L}_2(F)}{\partial o_2} = 0$, which is meaningless, since in this case the operations $f_1(\cdot), f_2(\cdot)$ are not distinguishable with the data points $\{(\mathcal{G}_k, y_k)\}_{k=1}^N$ for optimization.

Besides, since $o_1 = \frac{exp\{\alpha_1\}}{exp\{\alpha_1\}+exp\{\alpha_2\}}$, we have

$$\frac{\partial o_1}{\partial \alpha_1} = \frac{exp\{\alpha_1\}(exp\{\alpha_1\} + exp\{\alpha_2\}) - exp\{\alpha_1\} \cdot exp\{\alpha_1\}}{(exp\{\alpha_1\} + exp\{\alpha_2\})^2} \quad = \frac{exp\{\alpha_1 + \alpha_2\}}{(exp\{\alpha_1\} + exp\{\alpha_2\})^2} \tag{20}$$

Similarly,

$$\frac{\partial o_2}{\partial \alpha_2} = \frac{exp\{\alpha_1 + \alpha_2\}}{(exp\{\alpha_1\} + exp\{\alpha_2\})^2}. \tag{21}$$

Thus $\frac{\partial o_1}{\partial \alpha_1} = \frac{\partial o_2}{\partial \alpha_2}$. Then we have

$$\begin{gathered}\frac{\partial \mathcal{L}_2(F)}{\partial o_1} > \frac{\partial \mathcal{L}_2(F)}{\partial o_2} \\ \frac{\partial \mathcal{L}_2(F)}{\partial o_1}\frac{\partial o_1}{\partial \alpha_1} > \frac{\partial \mathcal{L}_2(F)}{\partial o_2}\frac{\partial o_2}{\partial \alpha_2} \\ \nabla_{\alpha_1}\mathcal{L}_2(F) > \nabla_{\alpha_2}\mathcal{L}_2(F).\end{gathered} \tag{22}$$

By Lemma A.1, o_1 decreases and o_2 increases. Then for the first task, we have

$$\begin{aligned}\mathcal{L}_1(F) &= \sum_k \Big(F(\mathcal{G}_k) - y_k\Big)^2 \\ &= \sum_k \Big(o_1 f_1(\mathcal{G}_k) + o_2 f_2(\mathcal{G}_k) - y_k\Big)^2 \\ &= \sum_k \Big(o_1 f_1(\mathcal{G}_k) + o_2 f_2(\mathcal{G}_k) - f_1(\mathcal{G}_k)\Big)^2 \\ &= \sum_k \Big(- o_2 f_1(\mathcal{G}_k) + o_2 f_2(\mathcal{G}_k)\Big)^2 \\ &= o_2^2 \sum_k \Big(f_2(\mathcal{G}_k) - f_1(\mathcal{G}_k)\Big)^2.\end{aligned} \tag{23}$$

The third equation utilizes the supposed ground-truth labeling function for the 1-st task, *i.e.*, $y_k = f_1(\mathcal{G}_k)$. Since $\sum_k \Big(f_2(\mathcal{G}_k) - f_1(\mathcal{G}_k)\Big)^2 > 0$ and o_2 increases, the loss of the first task $\mathcal{L}_1(F)$ increases. □

B. Implementation Details

B.1. Hyperparameters

For fair comparisons, all methods adopt the same dimensionality d as 512, number of layers as 2. Adam optimizer (Kingma & Ba, 2014) is adopted to optimize the model weights with a learning rate 1e-3 and another SGD optimizer with a learning rate 1e-2 is adopted to optimize architecture parameters for NAS methods. For our method, we adopt $K = 3$ for all datasets, and the hyperparameter $\lambda \in \{0.01, 0.1, 1, 10, 100\}$. We constrain the parameter size of the learned factors same to (Liu et al., 2023a)($\leq 1\%$ features), which is a replay-based method adopted by all the baselines for continual training. We adopt the dimensionality as $d_k = d/K$ for each module to keep the parameter size similar with the vanilla supernet for fair comparisons. We run the all experiments with 3 random seeds, and report the average performance and standard deviations.

B.2. Configurations

All experiments are conducted with:

- Operating System: Ubuntu 20.04.5 LTS
- CPU: Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz
- GPU: NVIDIA GeForce RTX 4090 with 24 GB of memory
- Software: Python 3.8.18, Cuda 12.2, PyTorch (Paszke et al., 2019) 2.1.2, PyTorch Geometric (Fey & Lenssen, 2019) 2.4.0.