Bilevel Optimization in the Deep Learning Era: Methods and Applications

Lei Zhang

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

November 14, 2022

Blacksburg, Virginia

Keywords: NAS, Graph, GNN, Architecture

Bilevel Optimization in the Deep Learning Era: Methods and Applications

Lei Zhang

(ABSTRACT)

Neural networks, coupled with their associated optimization algorithms, have demonstrated remarkable efficacy and versatility across an extensive array of tasks, encompassing image recognition, speech recognition, object detection, sentiment analysis, and more. The inherent strength of neural networks lies in their capability to autonomously learn intricate representations that map input data to corresponding output labels seamlessly. Nevertheless, not all tasks can be neatly encapsulated within the confines of an end-to-end learning paradigm. The complexity and diversity of real-world challenges necessitate innovative approaches that extend beyond conventional formulations. This calls for the exploration of specialized architectures and optimization strategies tailored to the unique intricacies of specific tasks, ensuring a more nuanced and effective solution to the myriad demands of diverse applications.

The bi-level optimization problem stands out as a distinctive form of optimization, characterized by the embedding or nesting of one problem within another. Its relevance persists significantly in the current era dominated by deep learning. A notable instance of its application in the realm of deep learning is observed in hyperparameter optimization. In the context of neural networks, the automatic training of weights through backpropagation represents a crucial aspect. However, certain hyperparameters, such as the learning rate (lr) and the number of layers, must be predetermined and cannot be optimized through the conventional

chain rule employed in backpropagation. This underscores the importance of bi-level optimization in addressing the intricate task of fine-tuning these hyperparameters to enhance the overall performance of deep learning models. The domain of deep learning presents a fertile ground for further exploration and discoveries in optimization. The untapped potential for refining hyperparameters and optimizing various aspects of neural network architectures highlights the ongoing opportunities for advancements and breakthroughs in this dynamic field.

Within this thesis, we delve into significant bi-level optimization challenges, applying these techniques to pertinent real-world tasks. Given that bi-level optimization entails dual layers of optimization, we explore scenarios where neural networks are present in the upper-level, the inner-level, or both. To be more specific, we systematically investigate four distinct tasks: optimizing neural networks towards optimizing neural networks, optimizing attractors towards optimizing neural networks, optimizing graph structures towards optimizing neural network performance, and optimizing architecture towards optimizing neural networks. For each of these tasks, we formulate the problems using the bi-level optimization approach mathematically, introducing more efficient optimization strategies. Furthermore, we meticulously evaluate the performance and efficiency of our proposed techniques. Importantly, our methodologies and insights transcend the realm of bi-level optimization, extending their applicability broadly to various deep learning models. The contributions made in this thesis offer valuable perspectives and tools for advancing optimization techniques in the broader landscape of deep learning.

Contents

Chapter 1

Introduction

1.1 Overview

Bi-level optimization stands out as a powerful and versatile technique with broad applicability, particularly in the dynamic landscape of deep learning. Its versatility is exemplified by its successful application across various neural network architectures, ranging from Convolutional Neural Networks (CNNs) to Graph Neural Networks (GNNs), amplifying its impact in the field. In the realm of computer vision, CNNs bi-level optimization can be applied to enhance the performance of CNNs by optimizing both the convolutional layers (lower level) and hyperparameters like learning rates or dropout rates (upper level). Bi-level optimization provides a systematic approach to fine-tune CNN architectures for specific image recognition tasks, leading to improved accuracy and efficiency. For GNNs, bi-level optimization proves invaluable in optimizing GNN architectures or graph structure, where the lower level involves learning graphs or training graph convolution kernels. The adaptability and effectiveness of bi-level optimization make it an indispensable tool in the deep learning era. Its applications span across diverse neural network architectures and sub-fields, reinforcing its significance in advancing the state-of-the-art in deep learning research and applications.

A traditional bi-level optimization challenge entails a nested problem structure, comprising an upper-level objective and a lower-level objective. Denoting the upper-level objective function as $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ and the lower-level objective function as $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, the formulation of the bilevel problem is expressed as:

$$\begin{aligned} &\min_{x_u \in X_U, x_l \in X_L} F(x_u, x_l), \\ s.t. \quad & x_l \in \arg\min_{x_l \in X_L} \{ f(x_u, x_l) : g_j(x_u, x_l) \leq 0, j = 1, \dots, J \}, \\ & G_k(x_u, x_l) \leq 0, k = 1, \dots, K, \end{aligned} \tag{1.1}$$

where the upper level constraints are denoted by $G_k : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ for $k = 1, \dots, K$, while the lower level constraints are represented by $g_j : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$. Any equality constraints, for the sake of brevity, have not been explicitly included. The sets $X_U \subset \mathbb{R}^n$ and $X_L \subset \mathbb{R}^m$ in this definition may indicate additional constraints such as integrality. Unless specified otherwise, it is commonly assumed that these sets refer to real numbers. The breadth of applications and optimization tasks encompassed by the definition can vary significantly based on the specific formulations of the upper-level objective function F and the lower-level objective function f.

Solving bi-level optimization problems presents a formidable challenge owing to their nested structures and the inherent high computational complexity, particularly in scenarios involving large neural networks. The intricacies arise from the interdependence of an upper-level optimization problem, which encompasses variables determined by a lower-level optimization problem. While traditional approaches have attempted to tackle this complexity, such as employing single-level reduction techniques or formulating surrogate problems that are ostensibly simpler to solve, these methods lack universality and may not offer a panacea for all instances. The application of single-level reduction involves simplifying the bi-level problem by transforming it into a single-level optimization problem. However, this strategy is contingent upon the nature of the specific problem and may not provide an optimal or generalizable solution across diverse scenarios. Additionally, proposing surrogate problems that mimic the original bi-level problem but are ostensibly easier to solve represents another avenue of approach. Nevertheless, this method faces challenges in accurately capturing the complexity and intricacies of the original problem, and its efficacy is subject to the specific characteristics of the problem at hand.

Despite these traditional approaches, the domain of bi-level optimization remains rife with unresolved challenges and open questions. The scalability and adaptability of these methods to complex scenarios involving extensive neural networks pose ongoing issues. The quest for a comprehensive and universally applicable solution persists, as the diversity of problems and the ever-evolving landscape of deep learning continue to unveil nuances that demand novel and nuanced strategies. In essence, while strides have been made in mitigating the computational complexities associated with bi-level optimization through conventional methods, the field remains a fertile ground for exploration. The need for innovative methodologies that can holistically address the intricacies of large neural networks within the context of bi-level optimization underscores the frontier nature of this research domain.

1.2 Outline

Within this dissertation, my emphasis is directed towards resolving four specific types of bi-level optimization problems, wherein either F or f in Eq. (1.1) serves as the objective of a neural network model. These encompass optimizing neural networks towards optimizing neural networks, optimizing attractors towards optimizing neural networks, optimizing graph structures towards optimizing neural network performance, and optimizing architecture towards optimizing neural networks.

In Chapter 2, we delve into a groundbreaking application of bi-level optimization employed in CNN models, referred to as gNAS. The concept underlying the gNAS framework we propose revolves around leveraging another neural network model to learn the distribution of neural architectures. It is adept at addressing the following challenges. 1) Difficulty in fully searching the neural architecture space efficiently without human heuristic space pruning. Due to the exponentially-large space of graph-structured neural architecture, NAS is an extremely challenging combinatorial search problem if without sufficient restriction by human heuristics. It is challenging to design an efficient method that captures large space with minimum prior knowledge restriction. 2) Lack of expressiveness in automatically capturing global characteristics of good architectures. Many current NAS methods incorrectly assume that the predetermined architectural distribution on each edge is not influenced by others [72]. These studies normally use adjacency matrix [46, 86] or path encoding [75, 76] which can only capture local architecture properties. Some recent methods such as RandWire [79] and NAGO [56] explored the space of classical network science models (such as random graphs), which are controlled by a small set of parameters that limits their expressive power in learning complex neural architecture patterns. 3) Difficulty in considering the high variance of wiring patterns in good architectures. The large variety of capable neural nets indicates that good architectures vary to some extent, and may be prone to perturbations [79], which may lead to a “good architecture by chance”. So instead of a deterministic single (sub-)optimal result, a “good-performing” family of similar architectures may provide better robustness.

In Chapter 3, we explore a widely studied task involving GNN and bilevel optimization, specifically, the learning of graph structures. Our proposed model addresses the challenge of automatically learning the causal graph for traffic prediction tasks using GNNs. Despite its significance, learning causal relationships in traffic data poses notable technical challenges: 1) Learning causal relationships on dynamic data: The dynamics of traffic data can be viewed as the outcome of underlying causal relationships. Tracing the causes of continuous dynamic data proves to be challenging. 2) The difficulty in simultaneously learning causal relationships and the autoregressive model: Inferring interpretable causal relationships while simultaneously enhancing the accuracy of the autoregressive model is a nontrivial task. 3) Model scalability on large spatiotemporal data: The number of entity-to-entity relationships to infer scales as $O(n^2)$ with the number of locations, presenting scalability challenges on extensive spatiotemporal datasets.

In Chapter 4, we delve into a compelling problem within a bi-level optimization context, where the inner problem is constrained to be an attractor or fixed point. In mathematical terms, a fixed point is an element that remains unchanged when subjected to a function. Our discovery reveals that by compelling a fixed point within a graph neural network, the network can be conceptualized as having an infinite number of layers, making it particularly adept at the graph-to-graph translation problem. The challenges include: 1) capturing long range node and edge dependencies at the same time, and 2) consistency between node attributes and edge attributes.

In Chapter 5, we tackle a traditional bi-level optimization challenge known as neural architecture search (NAS). In this context, the upper-level problem involves variables corresponding to the architectures of the inner neural network, while the inner-level problem pertains to the trainable layers of the neural network. However, our approach focuses on a unique task where the neural network takes the form of an Echo-State Network. We integrate Echo-State Network and NAS methodologies to address the translation problem from continuous node signals to dynamic graphs. The proposed method can jointly address the following challenges: 1) Difficulty in jointly extracting features from node dynamics while learning the dynamic relationships in a graph. The challenge necessitates that transformation patterns be considered in both time and graph dimensions. Moreover, these two dimensions are not independent, necessitating the need for a framework that can facilitate the combined evolution of node and edge dynamics. 2) Absence of an effective and scalable framework for graph dynamics encoding over a continuous long time duration with a high sampling rate. The inference of dynamic graph topology necessitates fine-grained, long-term knowledge on graph dynamics. Existing efforts for dynamic network embedding and representation learning are unable to efficiently manage extended time series of node attribute data. 3) Dilemma between learnability and efficiency of models. Modeling the complex mapping between node and edge dynamics demands models with a high capacity for learning. Compared to optimizing a model fitted to specific data, optimizing a highly flexible, highly learnable model is typically time-consuming.

Chapter 2

Bi-level Optimization in Generative Neural Architecture Search

2.1 Introduction

The study of neural architecture search (NAS) is intended to automate the design of neural network architectures with searching algorithms. Early NAS work used a fixed chain structure and only searched for the optimal sequence of operations along the chain [5, 98]. The basic ingredients are normally taken from hand-designed neural networks such as Inception [64], MobileNetV2 [58], MobileNeXT [95], and EfficientNet [66]. Based on the chain structure, a common practice is to search for architectures in a small cell and repeat the same cells in a chain [94, 98]. Later, DARTS was proposed for learning a small directed acyclic graph (DAG) in a cell [46], which enriches the neural network wiring topology. However, the small cell design limits the possible wiring patterns in the whole graph, while the way the cells are connected is also predefined. RandWire was among the first works that attempted to explore a more diverse set of connectivity patterns in neural network architectures and showed exciting results by simply wiring the basic operations randomly [79]. However, even RandWire tried to randomly wire networks, using a random graph model such as the Watts–Strogatz (WS) model is a strong prior knowledge, yet with limited model expressiveness in capturing network behaviors. But the inspiration from the above works indeed indicates the prospects to explore the diversity of wiring patterns, though it seems a highly under-explored domain with open questions such as: What are the optimal ways to model the network wirings? Is it possible to learn the wiring patterns? How do we further eliminate the priors on network wiring and search for novel wiring patterns?

Despite the prospects to answer the above questions, it is not easy due to several unsolved challenges: 1) **Difficulty in fully searching the neural architecture space efficiently without human heuristic space pruning.** Due to the exponentially-large space of graph-structured neural architecture, NAS is an extremely challenging combinatorial search problem if without sufficient restriction by human heuristics. It is challenging to design an efficient method that captures large space with minimum prior knowledge restriction. 2) **Lack of expressiveness in automatically capturing global characteristics of good architectures.** Many current NAS methods incorrectly assume that the predetermined architectural distribution on each edge is not influenced by others [72]. These studies normally use adjacency matrix [46, 86] or path encoding [75, 76] which can only capture local architecture properties. Some recent methods such as RandWire [79] and NAGO [56] explored the space of classical network science models (such as random graphs), which are controlled by a small set of parameters that limits their expressive power in learning complex neural architecture patterns. 3) **Difficulty in considering the high variance of wiring patterns in good architectures.** The large variety of capable neural nets indicates that good architectures vary to some extent, and may be prone to perturbations [79], which may lead to a "good architecture by chance". So instead of a deterministic single (sub-)optimal result, a "good-performing" family of similar architectures may provide better robustness. A more comprehensive related work section can be found in Appendix C.

To address the above challenges, we propose a novel generative neural architecture search (gNAS) framework that simultaneously learns the distribution of graph-structured neural architectures and identifies the “sweet regions” in the high-dimensional space much more efficiently than in original graph space. To tackle challenge 1, we employ the neural DAG representation and expand the search space to cover all possible wiring patterns. An MLP-based generator is presented to encode the distribution of architectures to overcome challenge 2. A formal expressiveness analysis is provided to demonstrate the great expressiveness of the generator. Lastly, the generator is designed to be stochastic in order to account for the variances of optimal structures when solving challenge 3. A smoothness analysis proves the feasibility of the optimization task. We conducted extensive experiments showing that gNAS generates competitive networks and achieves state-of-the-art NAS results on four datasets.

2.2 Problem Formulation

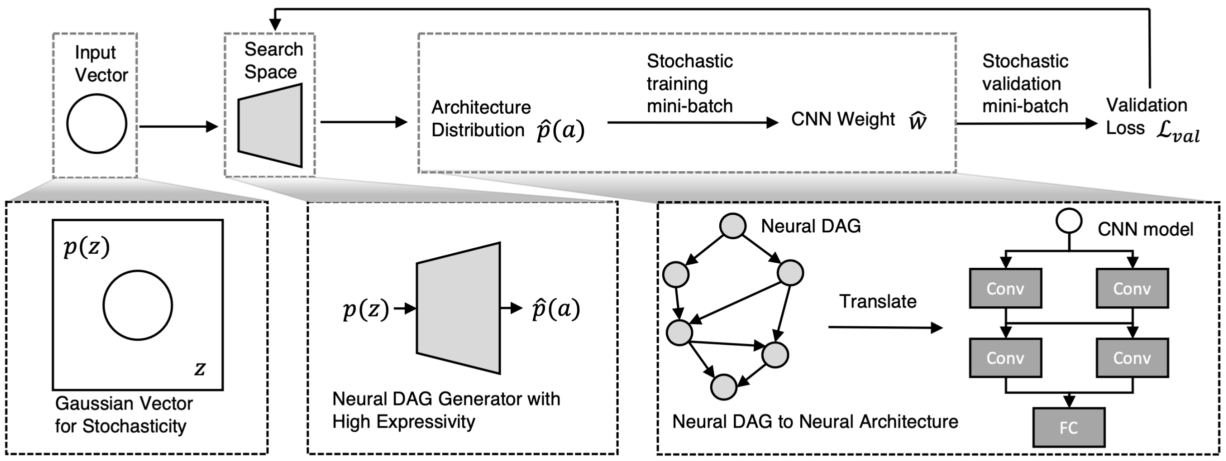
The study of NAS mainly covers two aspects, the search space and the search algorithm. The search space is a pool of options or hyperparameters for network architectures. Given a pre-defined search space $\mathcal{S}$, the search algorithm chooses the best combination $\mathcal{S} \rightarrow s$ such that $s \xrightarrow{\mathcal{K}} a$ where $\mathcal{K}$ is the prior knowledge that transforms the search result into neural network architectures, and a is an instance of network architectures.

The Problem of Generative Neural Architecture Search: Instead of searching for a single network, here we focus on searching for the distribution of networks with “good architectures”. We name it as the problem of generative neural architecture search (**gNAS**), which maps the neural architectures into latent space for searching for good architectures. The problem is more formally defined as follows: denote g as a mapping from a latent space to the space of neural network architectures, then we have $g : z \xrightarrow{\theta} a$ where z is a vector following distribution $p(z)$ and $a \in \mathcal{A}$ is a neural network architecture instance. The goal of gNAS is to learn the optimal mapping function g (parameterized with θ) that most probably generates good architectures.

gNAS is a new problem partially stimulated by the recent success of generative representation learning. The major problem of them is that the prescribed network models they normally use heavily rely on strong human priors and yet with small model expressiveness. For example, an Erdős-Rényi random graph model [36] only has two parameters, namely the number of nodes and the average edge degree, which tend to be far from enough to explain the complex behaviors of neural networks. Another challenge is the difficulty in jointly handling the tasks of finding g, the search for θ, and the learning of neural networks with candidate architectures. It is important to efficiently handle discrete values of graph-structured neural architecture and continuous values of latent semantic space, which will pose troubles to the gradient-based method. Additionally, there is no theoretical guarantee that the graph generator in graph-based NAS is capable of approximating the ideal network wiring solution, referred to as the graph generator's expressivity. Furthermore, there is no theoretical certainty that it is possible to optimize the graph generator's parameters to their ideal value.

2.3 Proposed Approach

This section outlines our proposed gNAS framework, which solves all of the issues raised above. Fig. 2.1 illustratively summarizes the whole generative neural architecture search process. In the gNAS framework, the search space is set to be the parameter space of a stochastic neural DAG generator that can generate different architectures that share same global characteristics. The results of gNAS are expected to be a class of neural network architectures, not a single one. This is inspired by the observation that many SOTA neural networks (e.g., EfficientNet[67], MobileNet[33], MobileNeXt[95], ResNet[31]) all have great



Input Vector
Search Space
Architecture Distribution $\hat{p}(a)$
Stochastic training mini-batch
CNN Weight $\hat{w}$
Stochastic validation mini-batch
Validation Loss $\mathcal{L}_{val}$
$p(z)$
z
Gaussian Vector for Stochasticity
$p(z)$
$\hat{p}(a)$
Neural DAG Generator with High Expressivity
Neural DAG
Translate
CNN model
Conv
Conv
Conv
Conv
FC
Neural DAG to Neural Architecture

Figure 2.1: Overview of Generative Neural Architecture Search (gNAS). The search space is in the parameter space rather than the embedding space. Given the random vector $z \sim p_\phi(z)$, the network generator can generate architectures following a certain distribution $p(a)$. The network generator's weights θ are optimized based on the validation loss $\mathcal{L}_{val}$.

performance but with totally different architectures. For more details, we start by introducing the search space that gNAS handles first.in Section 2.3.1, Tthen we formulate the gNAS network generator $g(\theta)$ and discuss its the theoretical guarantee in Section 2.3.2. At last, the optimization-based search stretegy is detailed in Section 2.3.3.

2.3.1 gNAS Search Space

The search space of the gNAS framework is defined as all the *wiring patterns* of neural networks. We characterize neural network architectures as neural directed acyclic graphs (DAGs), then define the parameter space of a neural DAG generator as the search space for all the wiring patterns in the neural DAG. In traditional NAS frameworks, the search space is directly defined as the architecture space, and NAS produces a single architecture. We argue that the best performing designs should be a class of architectures with similar hidden properties, as RandWire has partially demonstrated [79]. Defining the parameter space θ as the search space enables us to discover good global wiring patterns of neural networks. The gNAS search space is also more general than all the existing NAS space settings in terms of the wiring patterns. In particular, in contrast to previous work that imposes various limits on how neural networks can be wired, the gNAS search space encompasses all conceivable wiring patterns in a $n-$node graph with a maximum $n(n-1)/2$ edges

We assume that a deep neural network (e.g., a CNN) architecture can be defined as a neural DAG: $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = v_1, ..., v_n$ is the node set and $\mathcal{E} \in (v_i, v_j)|v_i, v_j \in V, j > i$ is the set of edges. Each node v_i in the neural DAG represents an operation such like convolution or pooling. Each edge represents the forward direction of the information from one node to another in the neural network. Without loss of generality, here in this paper we focus on common convolutional neural network architecture (CNN). The details are as follows:

Node Operation. The nodes in the neural DAG represent operations that can be chosen from depth-wise separable convolution, regular 3×3 convolution, shortcut, and so on. All the operations maintain the same number of channels as the input data flow by default.

Merging Operation. Edge $v_i \rightarrow v_j$ is defined as a direct data flow. Before multiple data flows arrive at the same node, the data flow will be merged. The merging stategy can be chosen from weighted sum, attention weighted sum, and concatenation. The number of channels remains the same after the merging.

Downsampling Downsampling refers to a reduction in the resolution of feature maps, wherein the number of channels either remains constant or increases [46, 64]. We separate the architectures into multiple segments (sub-graphs). The downsampling happens at the beginning of each segment.

Representing CNN architectures as neural DAGs helps us understanding the constraints in exiting neural network architectures and NAS search space. In DARTS, the cells are designed to have explicitly two input nodes and one output node, where normal cells and reduction cells are stacked in a chain. In existing work such as RandWire and NAGO, the neural DAG is constrained to be certain types of graph that are transformed following a pre-defined rule. None of existing work has the search space that can cover all possible neural DAGs.

2.3.2 Neural DAG Generator

Neural architecture search in the parameter space differs significantly from traditional NAS and requires a novel formulation. We define a neural DAG network generator as $g_\theta(z)$ where z is a $d-$dimensional random vector and θ is the parameter of the generator. When θ is fixed and z follows a distribution $p_\phi(z)$, we can generate a class of architectures by feeding sampled z into the architecture generator: $a = g_\theta(z), z \sim p_\phi(z)$. We assume that the parameter space of the neural DAG generator can encode important global characteristics of the optimal neural network architectures. The outcomes of the neural DAG generator are influenced not just by its weights but also by the random vector, making it a stochastic generator. Different from methods such as RandWire and NAGO, the stochastic neural DAG generator can be optimized using a gradient-based algorithm and can approximate any neural DAG distribution.

More specifically, we propose generative neural network to realize the stochastic neural DAG generator and prove its effectiveness on expressing neural DAG distributions. Given vector z and parameters θ, for a graph with maximum N nodes, the result of the neural DAG generator is $E = softmax(\mathrm{MLP}_\theta(z))$. $E \in \mathbb{R}^{\frac{n(n-1)}{2} \times N_{op}}$ where $\frac{n(n-1)}{2}$ is the maximum number of edges in a neural DAG and N_{op} is the number of possible operations on an edge. When we only consider the topology of the neural network regardless the operations, N_{op} can be set to 2. In this case the operations and the graph generation are completely decoupled. The existence of the edge $i \to j$ is determined by $E^{(i,j)} > \delta$ where δ, normally set as 0.5, is the threshold of keeping the edge. When $N_{op} > 2$, the neural DAG generator can also generate the specific operations on each edge given a set of all possible operations. In this case the operation on edge i is determined by $\arg\max(E_i)$.

Given a random vector with a simplistic isotropic Gaussian prior $\mathcal{N}(0, 1)$, we expect that the MLP graph generator can approximate another unknown distribution of the ideal architectures precisely. Firstly, the graph generator must have strong expressivity to cover as many network architecture distributions as possible. Then, the optimization process must guarantee that the graph generator converges to the optimal solution in the search space. The convergence of the optimization is discussed in the Section 2.3.4. For the expressiveness of the stochastic neural DAG generator, we have the following proofs.

Proposition 2.1. *Our neural DAG network generator can approximate the common graph generators including ER, BA, and WS used in existing work such as RandWire and NAGO with asymptotical zero error.*

To prove Proposition 1, we need Lemma 1 as follows:

Lemma 2.2. *If a generative model can be represented as the composition of n Barron functions, it can be approximated using a neural network with $n + 1$ layers.*

Lemma 2.2 provides expressively guarantee for neural networks on transforming a Gaussian distribution to another arbitrary distribution, whereas Proposition 2.1 requires proving the neural network's ability on transforming a Gaussian vector to neural DAG's distribution. A Barron function, as described in Lemma 2.2, is a function that can be effectively approximated by a neural network with a single hidden layer. The proof for Lemma 2.2 can be found in [43], shedding light on the capacity of multilayer perceptrons (MLPs) as generative models. Given the constraints of space, we utilize Lemma 2.2 to establish the validity of Proposition 2.1 in Appendix. Our Proposition 2.1 can be viewed as an extension of the classical Barron's Theorem [7], which provides a Fourier criterion for the approximability of a function by a neural network with a single hidden layer. We also discuss the expressively issues with the methods that are not using a generator (e.g., DARTS [46], IDARTS [90], MiLeNAS [30], PC-DARTS [81]) in Appendix A. To the best of our knowledge, it is the first NAS study that gives expressiveness guarantee for search space.

2.3.3 gNAS Optimization and Training

In the gNAS framework, the generator g's parameter θ can be seen as the hyper-parameter of the neural network. Without loss of generality, here we consider two edge types denoted as $E^{(i,j)} \in \{0, 1\}$ where 0 indicates that the edge does not exist and 1 indicates that it does. Suppose output of the r-th node in the neural network is $x^{(r)}$, given input data $x^{(0)}$ and labels y, the training loss of the network conditioned on the θ can be represented as:

$$\begin{aligned} &\mathcal{L}_{train}(x^{(0)}, y, W, \theta) = ||\hat{y} - y||^2 \quad where \quad \hat{y} = x^{(n)} \\ s.t. \quad &x^{(r)} = o[\sum_{i=1}^{r}(E^{(i,r)} x^{(i)})] \quad where \quad E = g_\theta(z) \end{aligned} \tag{2.1}$$

where o denotes the operation on the node (e.g., convolution, pooling), W denotes the weights of the neural network, $x^{(0)}$ and y denote the training data (for simplicity, we omit $x^{(0)}$ and y for following loss functions). The parameter to optimize in Eq 2.1 is θ. There are two challenges with optimizing θ via the loss $\mathcal{L}$. To begin, the edge set E is a discrete value that cannot be directly optimized using gradient-based techniques. Second, differentiation of optimization problem solutions is expensive. We present strategies based on continuous relaxation and stochastic approximation to address these two issues.

Continuous relaxation. We use the gradient-based optimization. Let $\mathbb{E}$ be a set of candidate operations (exist or not of an edge when $N_{op} = 2$, or various operations when $N_{op} >$

2). In order to render the search space continuous, we transform the categorical selection of a specific operation into a softmax distribution encompassing all potential operations:

$$x^{(r)} = o[\sum_{i=1}^{r}(\sum_{e\in\mathbb{E}} \frac{exp(E_e^{(i,j)})}{\sum_{o'\in\mathbb{O}} exp(E_{e'}^{(i,j)})} x^{(i)})] \tag{2.2}$$

$x^{(r)}$ is a weighted sum for all results of all possible edge operations. The weight for each edge type indicates the importance of the type. In the NAS stage, we use the mixed operation as the node operations. In the training stage, we replace the mixed operations as the selected operations.

Stochastic approximation After the relaxation of the discrete search space, our goal is to jointly learn the neural DAG generator model parameters θ and the CNN model parameters W, which is a bi-level optimization problem.

$$\begin{aligned} \min_{\theta} \quad & \mathbb{E}[\mathcal{L}_{val}(W^*(\theta), \theta)] \\ s.t. \quad & E = g_\theta(z), z \sim p_\phi(z) \\ & W^*(\theta) = \arg\min_{W} \mathcal{L}_{train}(W, \theta) \end{aligned} \tag{2.3}$$

Normally, both $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ are calculated as average loss across multiple mini-batches. We replace $\mathcal{L}_{train}$ with $\mathcal{L}'_{train}$ which is the loss evaluated on a random mini-batch from the training set. Similarly, We replace $\mathcal{L}_{val}$ with $\mathcal{L}'_{val}$ which is the loss evaluated on a random mini-batch from the validation set. The optimization using mini-batch-level loss can be considered as a stochastic approximation of the original problem. It been proven that this approximation leads to a sufficient descent direction, which leads the bi-level optimization converges to a stationary point [90]. The overall algorithm is given in Alg. 1. Lines 4-5 conducts stochastic approximation. The random vector z fed into the model as input (line

6) in each batch makes the neural DAG generation a stochastic process. The continuous relaxation makes the automatic differentiation (lines 7-10) possible. Note that the gradient methods in DARTS [46] and MiLeNAS [30] optimize the local architecture $E^{(i,j)}$ directly, while we optimize the weights of the stochastic generator $g_\theta(z)$ where z is a random vector.

Algorithm 1: Optimization Algorithm

```
 1: Initialize $W$, $\theta$
 2: for $e$ in epoch do
 3:     while not converged do
 4:         Sample mini-batch from the training data
 5:         Sample mini-batch from the validation data
 6:         Initialize random vector $z$
 7:         $E = g_\theta(z)$
 8:         Update $E$ by descending $\nabla_E \mathcal{L}'_{val}(W - \xi \nabla_W \mathcal{L}'_{train}(W, \theta), \theta)$
 9:         Update $\theta$ with the chain rule
10:         Update $W$ by descending $\nabla_W \mathcal{L}'_{train}(W, \theta)$
11:         Save the optimal generator weights $\theta$
12:     end while
13: end for
14: Evaluate on the searched neural network architecture.
```

2.3.4 Continuity Analysis

In contrast to the majority of NAS-related work, which focuses on optimizing the sole architecture of a neural network, we want to find a distribution of good architectures. A random vector is put into the architecture generator during the NAS process to generate the variations. Our formulation of the NAS indicates two merits: (1) given a random vector with Gaussian distribution, the architectures created are similar; and (2) the performance is robust against small change of architectures.

We use the concept of *Lipschitz continuity* to discuss these two merits. Proving the Lipschitz continuity guarantees that the function does not explode at some point. A smaller Lipschitz constant means that the function is smoother and easier to optimize. While calculating the

Lipschitz constant is extremely difficult and NP-hard [61], we can prove the existence of the upper bound, which rules out the possibility of the worst situation. To justify the first merit, we prove the Lipschitz continuity of the graph generator in respect to its input, the random vector in Proposition 2.3. For the second merit, we prove the Lipschitz continuity of a neural network's loss function in respect to its architecture in Proposition 2.4.

Proposition 2.3. *Our neural DAG generator is Lipschitz continuous.*

Proposition 2.4. *Given the training and validation data, the loss function of the DAG-based neural network is Lipschitz continuous with respect to the graph edges.*

The detailed proof for Proposition 2.3 and Proposition 2.4 are in the Appendix A due to the space limit. The framework illustrated in Figure 2.1 in an end-to-end optimization problem which involves optimizing both the neural DAG generator and the CNN model. In Proposition 2.3 we prove the continuity of the optimization from the random vector to the neural DAG. In Proposition 2.4 we prove the continuity of the optimization from the neural DAG to the training loss. Proposition 2.3 and 2.4 together give continuity guarantee of the whole framework.

2.4 Experiments and Results

Experiments were conducted using four image datasets: CIFAR10, CIFAR100, Flowers102, and ImageNet. The number of network parameters was restricted to 6.0M for ImageNet and 4.0M for the remaining datasets. For ImageNet, due to resource limitations, we follow the transfer learning scheme in DARTS[46] that uses the graph structure optimized on the CIFAR10 dataset.

Search space. We did not use the graph generator to wire a whole architecture due to

Table 2.1: Test accuracy (%) and search cost (GPU days) on CIFAR-10, CIFAR-100, and Flowers102 datasets. The top four methods are in the same RNAG* search space. The bottom three methods are in the same HNAG* search space.

	CIFAR10		CIAR100		Flowers 102	
	Accuracy	Cost	Accuracy	Cost	Accuracy	Cost
RandWire	94.1±0.16	-	71.7±1.34	-	94.7±0.46	-
RNAG-BOHB	94.3±0.13	14.0	73.0±0.50	14.0	95.7±0.38	11.8
RNAG-MOBO	94.0±0.26	-	71.8±0.50	-	94.9±0.53	-
gNAS+RNAG*	**94.8±0.63**	**2.5**	**74±0.70**	**2.5**	**96.2±0.49**	**2.5**
HNAG-BOHB	96.6±0.07	12.8	79.3±0.31	12.8	97.9±0.18	10.4
HNAG-MOBO	96.6±0.15	-	77.6±0.45	-	98.1±0.19	-
gNAS+HNAG*	**96.8±0.44**	**2.5**	**79.4±0.45**	**2.5**	**98.1±0.19**	**2.5**

the resource constraints. While the continuous relaxation in section 2.3.3 makes automatic differentiation possible, it requires large GPU memory especially when the graph is large. It is known that in the DARTS-like frameworks, all the possible edges and operations are explicitly considered within the NAS period, which results in high memory usage while the number of nodes increases [81]. We experimented with the following two search spaces:

1) **HNAG***: HNAG* is the modified version of the HNAG graph generator from NAGO[56]. HNAG is a three-level, hierarchical generator. At the highest level, there is a graph composed of cells, with each cell being depicted by a mid-level graph. Likewise, every node within a cell forms a graph comprising fundamental operations such as conv3×3, conv5×5, and so on. Different from HNAG where the search space is the parameters of the random graph generators, the search space of HNAG* is all possible graphs with n nodes. The number of nodes in each level is directly derived from the NAGO results. Taking CIFAR-10 as an example, NAGO suggests a 8-top level nodes, 1 middle-level node, and 5 bottom-level nodes graph. In this case, gNAS's search space is all the possible graph structures in the three-layer hierarchical graph, including $2^{[\frac{8\times(8-1)}{2}\times\frac{5\times(5-1)}{2}]}$ variants. The downsampling process for NAGO's HNAG search space happens at 1/3 and 2/3 nodes in the top layer of nodes. Please

note the number of nodes at each level is part of the NAS results in NAGO, so our approach cannot completely replace NAGO.

2) **RNAG***: RNAG* is a more general version of RandWire's architecture[79]. Similar with RandWire, RNAG* is also an architecture with three graphs connected in sequence, where each graph has 32 nodes. Different from RandWire, where the search space is the parameters of the random graph generators, the search space of RNAG* covers all possible graph distributions with 32 nodes. The downsampling process for the Randwire search space happens at the beginning of each graph.

In both search space settings, we loosen most wiring pattern constrains in their vanilla versions and utilize only the minimum human design (e.g., number of stages, number of nodes).

NAS settings. During the NAS phase, the batch size is set to 128 and the random vector is sampled for each batch. We train 50 epochs in total in the NAS period and use graph generator weights that achieve the highest validation performance. 50% of the training images are used as training set and 50% are used use the validation set.

Fully Training Configuration: Following the training of the graph generator, in the fully training phase, five instances of the network are randomly sampled for training, utilizing 100% of the training images. For all datasets, excluding ImageNet, we assess the performance of the generators trained on the combined training and validation sets, and then proceed to train them to completion (600 epochs). The initial learning rate is set to 0.025, with a batch size of 96. For ImageNet, we adhere to the complete training protocol of the small model regime, which involves training the networks for 250 epochs. The initial learning rate is set to 0.025, with a batch size of 256, a weight decay of 5e-5, and a momentum of 0.9. Cutout is applied with a length of 16 for small-image tasks and a size of 112 for large-image tasks.

Table 2.2: Average performance gain in % between the NAS-searched architectures and randomly-searched architectures.

Method	CIFAR10	CIFAR100	Flowers102
DARTS	0.32	0.22	0.25
PDARTS	0.52	1.99	0.2
NSGANET	-0.47	1.36	1.46
ENAS	0.01	-3.4	0.47
CNAS	0.73	-0.89	-2.48
MANAS	0.17	-0.2	0.69
StacNAS	0.42	2.86	-0.16
NAO	0.43	0.0	-0.13
NAGO+HNAG	0.94	2.85	2.93
NAGO+RNAG	0.21	1.81	1.05
gNAS+HNAG*	**1.14**	**2.98**	**2.93**
gNAS+RNAG*	**0.74**	**3.2**	**1.58**

Notably, DropPath, auxiliary towers, data augmentation, or other sophisticated training techniques are not employed in our experiments. All experiments were conducted using Nvidia Tesla V100 GPUs (details provided in the Appendix).

Results on Small Datasets

Table 2.1 shows the results of gNAS and comparison methods, i.e., NAGO and RandWire, under the same search space settings on three small datasets. It can be seen that in the RNAG* search space, gNAS always achieves better performance than NAGO and RandWire for all the datasets. In the HNAG* search space, gNAS achieves better performance than NAGO and RandWire for CIFAR10 and CIFAR100, achieves equal performance for Flowers102. As we use the gradient-based optimization algorithm, the search time of gNAS is much shorter than NAGO which utilizes Bayesian optimization (BO).

Table 2.2 shows the comparison of gNAS and other SOTA methods with different search space settings. We do this comparison to demonstrate the effectiveness of the NAS methods in their respective search space. For each NAS method, we calculate the average performance gain in percentage with and without using NAS in their search space. For architectures

Figure 2.2 shows the visualization of the comparison. We make the following observations:

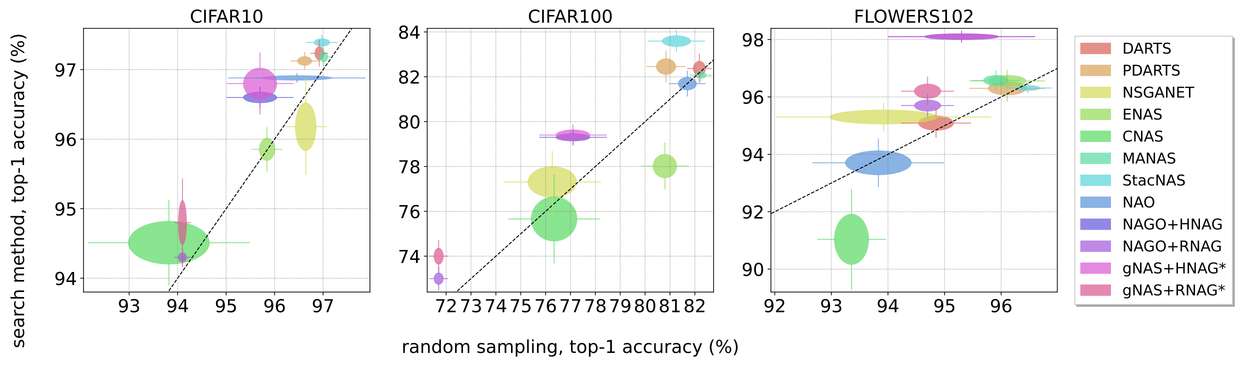


Figure 2.2: The chart illustrates the comparison of various NAS methods, with performance plotted on the y-axis, and random sampling results on the x-axis from their corresponding search spaces. The centers of the ellipses, along with their edges and whisker ends, depict the mean top-1 accuracy and standard deviation. Methods positioned along the diagonal indicate performance equivalent to the average architecture, while those positioned above the diagonal surpass it. All results, excluding ours (i.e., gNAS+HNAG*, gNAS+RNAG*), were sourced from public repositories provided by [84] and [56].

Table 2.3: Comparison with state-of-the-art image classifiers on ImageNet

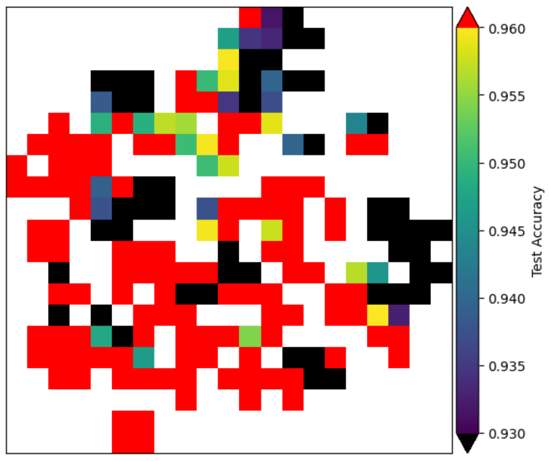
Method	Top 1 acc	Top 5 acc	Params (M)	Design Space	Search method
MobileNetV2	74.7	-	6.9	-	hand designed
ShuffleNetV2	74.9	92.2	7.4	-	hand designed
RandWire	74.7	92.2	5.6	RNAG*	hand designed
NASNet-A	74.0	91.6	5.3	NASNet	gradient-based
Amoeba-C	**75.7**	92.4	6.4	NASNet	evolutionary
DARTS	73.1	91.0	6.4	DARTS	gradient-based
NAGO+HNAG	**76.8**	93.4	5.7	HNAG*	BO
gNAS + RNAG*	**75.1**	92.3	5.9	RNAG*	gradient-based
gNAS + HNAG*	**76.8**	93.4	5.7	HNAG*	gradient-based

- The performance gain in the RNAG* search space is greater than on HNAG*. A reasonable explanation is that the number of nodes in the RNAG* search space in larger than in the HNAG* search space, so adjusting the network topology plays a more important role in determining the model's performance.

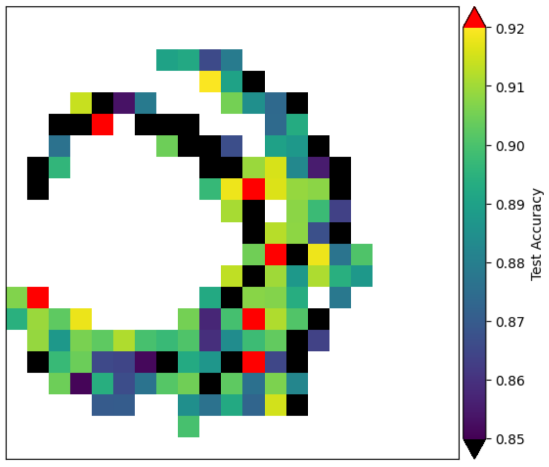
- The performance of gNAS in both search spaces shows a relatively high variance but the lowest performance is still decent. It means that the graph generator does learn the knowledge on how to better wire the networks to achieve better performance.

- gNAS + HNAG* achieves the highest improvement margin compared with doing random search in the same search space. It means that the good performance of the model does not come from a deliberately designed search space, but from the NAS process itself.

It is important to note that the best testing performance is just one way to evaluate a NAS framework; it is not the gold standard for NAS frameworks, as testing performance is heavily dependent on the search space and training tricks. Table 2.2 shows that some NAS frameworks obtain negligible or even negative performance gains in their search space when compared to randomly generated architectures. gNAS, on the other hand, gets the biggest performance improvement when compared to randomly generated architectures in its search space.



(a) Heat map of gNAS



(b) Heat map of D-VAE

Figure 2.3: Accuracy heat map comparison between gNAS and the VAE-based method

To better illustrate the Lipschitz continuity properties of gNAS as discussed in Proposition 2.3, 2.4, and the corresponding proofs in the Appendix, we visualize the accuracy heat map of gNAS an compare it with the VAE-based method D-VAE [91] which also has a hidden space. For each of the two methods, we randomly sample 1000 networks and apply t-SNE on the hidden space of z for dimensionality reduction. As shown in Fig. 2.3, we have the following observations: 1) The points in the gNAS hidden state space is much more evenly distributed than in the D-VAE, and 2) The loss surface of gNAS is much smoother than D-VAE. The visualizations are consistent with our assumption: instead of doing the search in a known unsmooth space, which most existing NAS works have, it is much easier to learn an neural DAG generator that can generate dense good architectures.

Results on ImageNet

Table 2.3 illustrates a comparison of results on ImageNet data. With a top-1 accuracy of 75.1 percent, gNAS + RNAG* exceeds the best RandWire model with WS (4, 0.75). In the HNAG* search space, gNAS has a mean top-1 accuracy of 76.8%, which is the same as NAGO-HNAG. We highlight that the gNAS generator was trained on CIFAR-10 and then transferred to ImageNet, so the generator does not overfit the ImageNet dataset. The search cost of gNAS is substantially lower than its competitors, and the re-trained models with generated architectures have better or comparable performance.

2.5 Discussion

Due to the high computation cost, in this study we do not search for node operations. Instead, we only optimize the neural network topology when the operations are fixed, similar with existing work such as NAGO [56] and RandWire [79]. As a result, even gNAS outperforms NAGO and RandWire in their search space, gNAS does not perform better than several other NAS methods on some datasets with different search space settings. The computation costs and GPU memory usage of gNAS grow quadratically to the number of nodes in the neural DAG. This is a common issue with gradient-based NAS frameworks such as DARTS [46] and MileNAS [30]. A single NVIDIA V100 with 32GB RAM can support the training of gNAS in both NAGO's and RandWire's search spaces.

We presented gNAS, a novel NAS framework that optimizes the stochastic neural DAG generator instead of the architecture of a single network. Our hypothesis is that numerous optimal architectures which have same global wiring characteristics exist. The neural DAG generator is used to encode such global characteristics while the random vector makes the generated architectures different under the same distribution. By encoding DAGs directly with the neural DAG generator, we're also able to generate novel architectures with the least human bias. Our work suggests a new transition from searching for an individual network to optimizing the parameters of a stochastic network generator, analogous to how deep learning replaced feature engineering in machine learning.

Chapter 3

Bi-level Optimizing in Graph Structure Learning for Traffic Prediction

3.1 Introduction

The prediction of highway traffic has been a longstanding focus for both industry and academia, representing a quintessential spatial-temporal data mining challenge. The development and implementation of an Intelligent Transportation System (ITS) hinge crucially on obtaining reliable, accurate, and consistently updated real-time traffic information. Integral subsystems of an ITS, including the Advanced Traveler Information System (ATIS) and the Advanced Traffic Management System (ATMS), heavily rely on high-quality real-time traffic data to furnish road users with current advisories and to enact effective traffic control strategies. Historically, the primary objective was the collection of real-time data, but there has been a recent shift in perspective among many agencies. They are now exploring the potential of leveraging extensive archived datasets for "real-time forward-looking analysis." With predictive data, proactive transportation management becomes a viable option. For instance, adaptive traffic signal control proves more effective when grounded in predicted traffic volume.

A general guideline for most existing traffic prediction models can be summarized by Tobler's First Law (TFL), which says that "Everything is related to everything else, but things that are nearby are more related than distant things." TFL indicates the importance of bridging the spatial dependency and the importance of the statistical spatial correlations between different locations. Most existing traffic prediction methods follow TFL by representing the spatial dependency with a road network or the Euclidean distance. However, road networks and Euclidean distances are not accurate for many real-world circumstances. For example, two nearby traffic sensors may be positioned in two lanes of traffic moving in opposite directions on the highway, and thus have quite a very low spatial dependency. Some other new methods learn the correlations between locations with history data and achieve better results. However, these methods suffer from low interpretability. A moderate correlation between two locations could be the result of confounding variables or biased data. Inspired by the work in causal inference in recent years[9, 41], we propose solving the traffic prediction task by considering the Granger causal relationship among traffic sensors. Granger causality is a classical statistical concept of causality that is based on prediction [24]. By explicitly modeling the Granger causal relationship between the traffic sensors, the model becomes more stable and interpretable, which is highly desirable in academia and industry. Modeling the causal relationship in traffic prediction tasks is beyond the scope of TFL and has not been studied yet.

Despite its importance, however, learning causal relationship in traffic data involves significant technical challenges: 1) **Learning causal relationships on dynamic data.** The dynamics of traffic data can be considered as the result of the ground truth causal relationships. Tracing the causes of continuous dynamic data is difficult. 2) **The difficulty in learning causal relationships and the autoregressive model simultaneously.** Inferring interpretable causal relationships while trying to improve the accuracy of the autoregressive model is nontrivial. 3) **Model scalability on large spatiotemporal data**. The number of entity-to-entity relationships to infer is $O(n^2)$ of the number of locations.

We present a novel Graph Neural Network (GNN)-based Spatiotemporal Causal Graph Inference (ST-CGI) framework to address the above challenges. This framework infers the Granger causality graph for all locations while optimizing the autoregressive model, which depends on the causality graph. This study's significant contributions are as follows: 1) the design of a novel framework for both traffic prediction and traffic causal inference; 2) proposal for a more efficient approximation of Granger causality; 3) Extensive experiments for both performance and interpretability.

The rest of this paper is organized as follows: Section 2 reviews the background and related work. Section 3 introduces the preliminaries. Section 4 presents all the components of our ST-CGI framework. The experiments on real-world data are presented in Section 5, and the paper concludes with a summary of the research in Section 6.

3.2 Problem Formulation

We target on the spatial temporal traffic forecasting problem by following the same formulation as previous work [45, 78, 92]. Consider an array of traffic series comprising N correlated univariate time series denoted as $X = X_{0,-}, X_{1,-}, ..., X_{t,-}, ...,$ where $X_{t,-} = X_{t,1}, X_{t,2}, ..., X_{t,N}{}^T \in \mathbb{R}^{N\times 1}$ represents the traffic data from N traffic sensors at time step t. Our objective is to predict future values of the traffic time series based on observed historical values. We frame this challenge as identifying a function g to forecast future traffic flow at the next step, utilizing the past S steps of historical data:

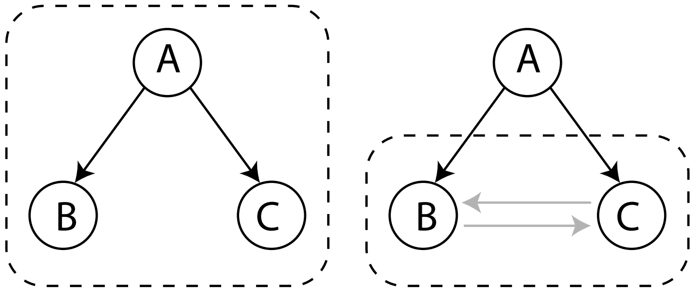
$$X_{t+1,-} = g(X_{[t-S:t],-}) \tag{3.1}$$

Definition 3.1. Granger Causality: A time series $X_{t,i}$ is Granger causal of another time series $X_{t,j}$ if including the history of $X_{-,i}$ improves prediction of $X_{-,j}$ over knowing of the history of $X_{-,j}$ alone. Specifically, this is quantified by comparing the prediction error variances of the one-step linear predictor, $\widehat{X}_{t,j}$, under two different models, the **restricted model** and the **unrestricted model**. The unrestricted model $g_u(X_{[t-S:t],-})_j = \widehat{X}_{t+1,j}$ uses the full histories of all the time series for prediction. The restricted model $g_r(X_{[t-S:t],-i})_j = \widehat{X}_{t+1,j}$ omits the putatively causal time series from the set of predictive time series. $X_{-,i}$ Granger causes $X_{-,j}$ if

$$\begin{aligned} &var(|X_{t+1,j} - g_u(X_{[t-S:t],-})_j| \mid X_{[t-S:t],-}) > \\ &var(|X_{t+1,j} - g_r(X_{[t-S:t],-i})_j| \mid X_{[t-S:t],-i}) \end{aligned} \tag{3.2}$$

here $var(|X_{t+1,j} - g_u(X_{[t-S:t],-i})_j| \mid X_{[t-S:t],-i}$ denotes the prediction performance of model g_u given histories data on all the nodes except for node i, i.e., $X_{[t-S:t],-i}$.

Granger introduced a statistical concept of causality grounded in the principles that (i) a cause precedes its effect, and (ii) understanding a cause enhances the prediction of its effect. However, a determination of Granger causality does not guarantee true causality. The Granger causality tests fulfill only the Humean definition of causality, which identifies cause-and-effect relations as those having constant conjunctions. If a common third process drives both X and Y with different lags, one may still fail to reject the alternative hypothesis of Granger causality. Figure (3.1) shows two cases of causal inference. In Case 1, node A is the confounder of both nodes B and node C. If all three nodes are all included in the data, even though node B and node C are correlated, Granger causality can identify that node B



(a) Case 1 (b) Case 2

Figure 3.1: Granger causality can identify confounders (Case 1) but cannot identify hidden confounders (Case 2).

and C do not have a causal relationship. However, in Case 2, when the confounder is not included in the observational data, Granger causality cannot discover the hidden confounding node, A. As a result, Granger causality may infer an inaccurate causal relationship between nodes B and C. Each of the components will be introduced in the following sections.

While Granger causality is a potential method for causal discovery, it depends highly on the selection of a multivariate autoregressive function. In traditional Granger causality methods, the multivariate autoregressive function is typically a linear model that cannot capture nonlinear interactions between entities. Furthermore, Granger causality requires running the autoregressive function multiple times to determine the marginal predictability when including and excluding different variables, which is complicated and time-consuming. To resolve all of these issues, we propose to build an end-to-end neural network model.

The Granger causality inference problem can be formulated as the following optimization problem:

$$\begin{aligned} G_C &= \arg\min_G \Big(\sum_{t=S}^{\mathcal{T}} |X_{t+1} - g_{W^*}(G, X_{[t-S:t]})| + \lambda\mathcal{R}(G)\Big) \\ \textit{where } W^*(G) &= \arg\min_W \sum_{t=S}^{\mathcal{T}} |X_{t+1} - g_W(G, X_{[t-S:t]})| \end{aligned} \tag{3.3}$$

where g is the neural network model parameterized with W. $\mathcal{R}$ is the regularization term on the causal graph, which enforces fewer edges in the graph. A concrete solution for g is introduced in the following sections.

The autoregressive function g takes the time series data and causal graph as input. When g only considers one-hop neighbors in G, the conditional log-likelihood given the embedding of nodes for predicting the graph signal $LL(X_{-,j}|e_{i,j})$ is equal to $LL(X_{-,-}|e_{i,j})$ because edge $e_{i,j}$ can only affect the prediction for node j. As a result, optimizing (3.3) guarantees that each edge $e_{i,j}$ in G indicates the Granger causality from node i to node j.

There are two terms in the outer optimization. The first term optimizes for the best autoregressive function G, which depends on the causal graph G. The second term $\lambda\mathcal{R}(G)$ penalizes the number of edges in the graph. As a result, if the edge does not contribute to the first term's prediction, the edge will be removed in the outer optimization problem. According to the analysis above, the problem in 3.3 is a bi-level optimization problem.

3.3 Proposed Approach

3.3.1 ST-CGI Framework

The bi-level optimization problem in equation (3.3) is expensive to solve. For every causal relationship $i \to j, i \neq j$, we need to build a restricted model and an unrestricted model.

Therefore, the computational complexity is at least $O(n^2)$. The causal graph we are trying to learn is a relatively stable relationship $\left[X_{(t-S):t}\right] \xrightarrow{G} X_{t+1}$. According to Norbert Wiener, whose work Granger cauaslity work built upon, "if a time series X causes a time series Y, then past values of X should contain information that help predict Y above and beyond the information contained in past values of Y alone". That inspired us to decouple the bi-level optimization problem as a multi-goal optimization problem that can be solved iteratively. The causal graph could be approximated with the relationship $\left[X_{(t-S):t}\right] \xrightarrow{G'} X_t$. Instead of optimizing on the causal graph directly, we propose to optimize the weight of a GNN that generates the causal graph. This approximation can be represented as a single-layer optimization problem in the following equation (3.4):

$$\begin{aligned} &\arg\min_{\Theta,W}\Big(\sum_{t=S}^{\mathcal{T}} |X_{t+1} - g(W, G^*, X_{[t-S:t]})| + \lambda\mathcal{R}(G_t^*)\Big) \\ &G_t^* = CGI(\Theta, X_{[t-S:t]}) \end{aligned} \tag{3.4}$$

where CGI is the causal graph estimator parameterized with Θ, and $\mathcal{R}$ is the L_1 regularization of the causal graph.

We propose to solve the causal inference and the autoregressive tasks simultaneously in a single end-to-end framework. Both the causal graph and the prediction for the next timestamp are inferred from the input data. The prediction for the next timestamp is dependent on the inferred causal graph. According to this requirement, we propose a framework, shown in Figure (3.2), with three modules trained jointly, namely a time series encoder, e, a causal graph estimator, and a causal graph-based autoregressive module.

The time series encoder e learns the high-dimensional representations of the multivariate input time series data. As shown in equation (3.5), the input data from P nodes $X \in \mathbb{R}^{P\times D\times S}$ are transformed into an embedding representation $h \in \mathbb{R}^{P\times R}$, where D is the input dimension

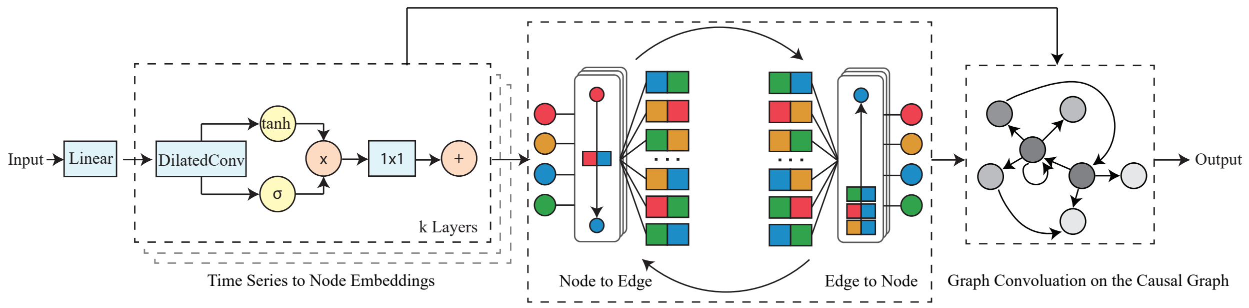


Figure 3.2: ST-CGI framework: The left module is the TCN-based time series encoder. The middle module is the deep graph neural network-based causal graph estimator. The right module is the single layer GCN for traffic forecasting. The traffic forecasting is based on both the traffic data embedding from the left module and the inferred causal graph from the middle module.

and R is the hidden dimension.

$$h_{t-S:t} = e(X_{t-S:t}) \tag{3.5}$$

As shown in equation (3.6), the causal graph estimator f learns the affinity matrix of the causal graph $A \in \mathbb{R}^{P\times P}$ from the embedding.

$$A_{t-S,t} = f(h_{t-S:t}) \tag{3.6}$$

Given the causal graph, the autoregressive model g tries to approximate the underlying linear/nonlinear mapping from historical traffic to future traffic.

$$\widehat{X}_{t+1} = g(A_{t-S,t}, X_{t-S:t}) \tag{3.7}$$

The model's overall formulation can be written as equation (3.8).

$$\begin{aligned}\widehat{X}_{t+1} =& \mathcal{F}_\theta(X_{t-S:t}) \\ =& g(f(e(X_{t-S:t})), X_{t-S:t})\end{aligned} \tag{3.8}$$

Given a long period of historical time series data and the problem formulation, we optimize the function in equation (3.9)

$$\begin{aligned} f, g, e = \arg\min_{f,g,e} \mathcal{L}(f, g, e, X_{train}) \\ \text{where } \mathcal{L}(f, g, e, X_{train}) = \\ \frac{1}{S\mathcal{T}} \sum_{s=1}^{S} \sum_{t=S}^{\mathcal{T}} (X_{t+1,s} - g(f(e(X_{t-S:t})), X_{t-S:t})) \end{aligned} \tag{3.9}$$

3.3.2 Causal Convolution for Time Series Encoding

The left module in Figure 3.2 demonstrates the structure of the time series encoder. Given raw multivariate time series data with lag S, the first step is to encode the time series into unified embeddings. The embedding should be a condensed vector that contains information about the temporal trends, temporal dependency, and temporal delays of the time series. The embedding must retain as much information as possible about the causal relationships between nodes.

Instead of using RNN-based units (e.g., LSTM, GRU), we opt to use dilated causal convolution network as our temporal convolution network (TCN) for encoding the time series data[69]. Causal convolution networks with dilation enable an exponentially expanded receptive field as the layer depth grows.

As shown in equation 3.10, gated dilated causal convolution [34] is utilized on every layer of the module. Because the model can become very deep when the input time series data is long, skip connections [31] are also used for each layer of convolutions. This way, the residual functions can be more easily learned during the optimization.

$$h = tanh(W_{f,k} * x) \odot \sigma(W_{g,k} * x) \tag{3.10}$$

Here, W represents the trainable parameter, k is the layer index, f and g refer to the filter and gate, $*$ stands for the convolution operator, $\odot$ signifies element-wise multiplication, and σ denotes a sigmoid function.

3.3.3 Causal Graph Estimator

After the time series on each location/node has been encoded as embeddings, we assume the embeddings contain essential information about the dynamics of the time series. The next step is to infer the causal relationship between the embeddings. As the causal relationship lies in a discrete space, GNN models are suitable for inferring the causal graph. GNNs have been shown to capture the relational connectivity between nodes [39].

It is natural to assume that the causal graph is related to a transportation road network in the spatial environment. For example, locations in closer proximity to one another have a higher chance of impacting one another. Different from the NRI model [39], we applied GNN models on the existing network structure rather than the complete graph.

There are several benefits of starting from an existing graph. First, the computational costs can be greatly reduced. Second, the GNN model converges faster. By stacking k GNN layers, the GNN model's receptive field is k-hops over neighbors in the existing graph, which is sufficient for inferring most of the causal relationships. In other words, the formula in (3.6) can be changed to $A_{t-S,t} = f(A', h_{t-S:t})$ where A' is the affinity matrix of the existing graph and f is a GNN model. Given the traffic flow embeddings h from the time series encoder in Eq. (3.10), the GNN model computes the following node to edge ($v \to e$) and edge to node ($e \to v$) message passing operations:

$$[]input : h_v^{(0)} = h \tag{3.11}$$

$$v \to e : h_e^{(m)'} = \sigma(W_e^{(m)} \begin{bmatrix} h_v^{(m)} R \\ h_v^{(m)} S. \\ h_e^{(m-1)} \end{bmatrix} + b_e^{(m)}) \tag{3.12}$$

$$e \to v : h_v^{(m)'} = \sigma(W_v^{(m)} \begin{bmatrix} h_e^{(m)} R^T \\ h_v^{(m-1)} \end{bmatrix} + b_v^{(m)}) \tag{3.13}$$

$$output : A = f_{out}(h_e^{(n)}) \tag{3.14}$$

where $h_v^{(m)}$ is the m-th layer node embedding, $h_e^{(m)}$ is the m-th layer edge embedding, f_{out} is the read-out function that transforms the last layer of the edge embeddings into the affinity matrix of the causal graph. R and S are the receiver and sender matrices of edges (detailed in the Appendix)

The middle module in Figure 3.2 demonstrates the structure of the causal graph estimator. The process of passing messages from nodes to edges and from edges to nodes can be iterated multiple times, thereby expanding the receptive field of the graph neural network.

3.3.4 Causal Graph Based Autoregressive Model

After the causal graph is inferred, we apply only one layer of GCN on the temporal embeddings of the input. The reason for this is that we focus on only inferring direct Granger causal relations rather than indirect causal relations. Using a single-layer GCN in the autoregressive stage enforces the causal graph estimator to learn a Granger causal graph. Given the traffic embeddings from the time series encoder and the affinity matrix inferred from the causal graph estimator, the final forecasting step

$$X_{t+1} = \sigma(AhW) \tag{3.15}$$

where h is the embedding representation of nodes learned from Eq. (3.5), and W is the parameter to optimize in this step.

Relationship with Other Models

Our proposed method is totally different from those who only used pre-defined graph structure (e.g., road networks, distance-based graphs, similarity graphs) for modeling the spatial dependency. Such methods include DCRNN[45], GEML [73], TGC-LSTM [15], ST-MGCN[22], STG2Seq[4]. The existing or pre-defined graph structures are helpful for learning the spatial dependency. However, it is inevitable that such graphs are noisy or miss certain information for different applications.

The most similar methods to our work are those that try to learn/optimize a graph structure for modeling the spatial dependency. Such methods include ASTGCN [26], AGCRN [3], GMAN [93], Graph WaveNet [78], and SLCNN [92]. Compared with these studies, our work has several unique features. Firstly, our framework is based on the Granger causality theory, which formulates the causality relationship, while the others depend on correlation relationships that are difficult to interpret. Secondly, the causal relationship we learn in the ST-CGI framework is directed, while most of the other methods regard the spatial dependency from a to b and from b to a are the same. Thirdly, ST-CGI decouples the spatial dependency modeling and the traffic data autoregressive task, which makes the model easy to interpret.

3.4 Experiments and Results

In this section, we introduce the experiment settings, performance comparison, and interpretability illustration.

Datasets

We assess the performance of ST-CGI on two publicly available traffic network datasets: METR-LA and PEMS-BAY. METR-LA encompasses four months of traffic speed statistics obtained from 207 sensors located on the highways of Los Angeles County. PEMS-BAY consists of six months of traffic speed information collected from 325 sensors in the Bay area. The data undergo the same preprocessing steps as outlined in [78]. Specifically, the raw time series data are aggregated into five-minute windows, followed by Z-score normalization. The datasets are then partitioned chronologically, with 70% allocated for training, 10% for validation, and 20% for testing.

Baseline Methods

SLCNN is compared with a traditional method, Auto-Regressive Integrated Moving Average (ARIMA), and the state-of-the-art methods including DCRNN[45], STGCN[89], SLCNN[92], and Graph Wavenet (GWN)[78].

Experiment Settings

In the time series encoder, we adopt a sequence of dilation factors (1, 2, 1, 2, 1, 2, 1, 2), a configuration akin to [78]. Furthermore, a dropout with a probability of 0.3 is employed on the outputs of the time series encoder. For the causal graph estimator, We apply a sequence of $v \rightarrow e, e \rightarrow v, v \rightarrow e, e \rightarrow v, v \rightarrow e$ message passing GNN operations. The readout function f_{out} in equation (3.11) is set to be two dense layers. The initial graph is the same distance-based graph used in [45, 78]. We train our model using Adam optimization algorithm with an initial learning rate of 0.001.

Performance Comparison

Following the same settings used by DCRNN, GWN, and SLCNN, the forecasting tasks are set in three levels, including 15 minutes, 30 minutes, and 1 hour ahead of forecasting. Table 3.1 and Table 3.2 show the comparison of different approaches for short term (15 minutes) traffic forecasting on both datasets. Table 3.3 and Table 3.4 show the comparison of different approaches for long term (30/60 minutes) traffic forecasting on both datasets. All methods undergo evaluation using three standard metrics in traffic forecasting tasks: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). The ensuing observations are as follows.

(1) For short term (15 minutes) traffic prediction, our proposed ST-CGI method achieves overall the best performance among all the methods.

(2) For long term (30/60 minutes) traffic prediction, GWN and SLCNN have very competitive performance. On METR-LA data, ST-CGI performs better than GWN but fails to beat SLCNN. On PEMS-BAY data, ST-CGI performs better than SLCNN but fails to beat GWN. The performance difference between ST-CGI and the best performed models is minor.

It is noteworthy that ARIMA is the only interpretable method among the baseline methods,

Table 3.1: Performance comparison for 15 minutes traffic prediction on the METR-LA dataset

Model	METR-LA (15 min)		
	MAE	RMSE	MAPE
ARIMA	3.99	8.21	9.6%
DCRNN	2.77	5.38	7.3%
STGCN	2.87	5.54	7.4%
SLCNN	**2.53**	5.18	6.7%
GWN	2.69	5.15	6.9%
ST-CGI	2.60	**5.14**	**6.7%**

Table 3.2: Performance comparison for 15 minutes traffic prediction on the PeMS-BAY dataset

Model	PeMS-BAY (15 min)		
	MAE	RMSE	MAPE
ARIMA	1.62	3.30	3.5%
DCRNN	1.38	2.95	2.9%
STGCN	1.46	3.01	2.9%
SLCNN	1.44	2.90	3.0%
GWN	1.30	2.74	2.7%
SG-CGI	**1.29**	**2.69**	**2.7%**

as ARIMA can be considered as a linear method. All of the rest baseline methods lack a theoretical foundation for interpretation. Our proposed ST-CGI method not only outperforms the only interpretable method but also performs better or on par with the non-interpretable deep neural network methods.

Interpretability Investigation

The theoretical foundations in causal inference enable ST-CGI great power in interpretability. After the model is trained, the first two modules in the framework can estimate the causal graph of given data. Causal graphs are directed graphs where an edge from node i to node j indicates the Granger causal relationship from i to j. In the traffic forecasting task, an edge with a high weight from traffic sensor i to traffic sensor j means that the traffic flow

Table 3.3: Performance comparison for 30/60 minutes traffic prediction on the METR-LA dataset

Model	METR-LA (30/60 min)		
	MAE	RMSE	MAPE
ARIMA	5.15/6.90	10.45/13.23	12.7%/17.4%
DCRNN	3.15/3.60	6.45/7.60	8.8%/10.5%
STGCN	3.48/4.45	6.84/8.41	9.4%/11.8%
SLCNN	**2.88/3.30**	**6.15/7.20**	**8.0%/9.7%**
GWN	3.07/3.53	6.22/7.37	8.4%/10.0%
ST-CGI	3.01/3.44	6.19/7.28	8.3%/9.8%

Table 3.4: Performance comparison for 30/60 minutes traffic prediction on the PeMS-BAY dataset

Model	PeMS-BAY (30/60 min)		
	MAE	RMSE	MAPE
ARIMA	2.33/3.38	4.76/6.50	5.4%/8.3%
DCRNN	1.74/2.07	3.97/4.74	3.9%/4.9%
STGCN	2.00/2.67	4.31/5.73	4.1%/5.4%
SLCNN	1.72/2.03	3.81/4.53	3.9%/4.8%
GWN	**1.63/1.95**	**3.70/4.52**	**3.7%/4.6%**
SG-CGI	1.68/2.01	3.77/4.65	3.8%/4.8%

at location j is heavily affected by the traffic flow at location i.

Figure 3.3 shows a causal graph inferred by ST-CGI on the METR-LA dataset. Despite that the directed edges are difficult to spot due to a large number of nodes, we can still get some information from the Figure. The first observation is that there are some "hubs" with large numbers of outgoing edges. This means that the traffic flow in this area is mainly influenced by a small number of locations.

One important observation is that the Granger causal relationship between two adjacent locations is usually not strong, which appears counterintuitive. According to the inferred causal graphs, a distant hub may have a greater impact on traffic flow in one location than the closest traffic sensor on the same road. It makes sense because the nature of Granger causality is to eliminate the indirect causes which have no incremental benefit on the predictions. In a real-world scenario, the traffic flow 15 minutes later near one traffic sensor should come from locations that are further than its closest traffic sensors.

3.5 Discussion

In this study, we propose a novel interpretable model (ST-CGI) for traffic prediction tasks based on Granger causality. Specifically, we formulate the traffic prediction task a bi-level optimization task with a causal inference module and an autoregressive module, where the latter module depends on the former module. The model does not only make predictions of the traffic flow but also provides evidence of the predictions which are based on. Experiments on two real-world datasets show that ST-CGI achieves state-of-the-art results and has advantages in making shorter-term predictions. The interpretability and visualization function of ST-CGI is highly desired by domain experts and decision-makers.

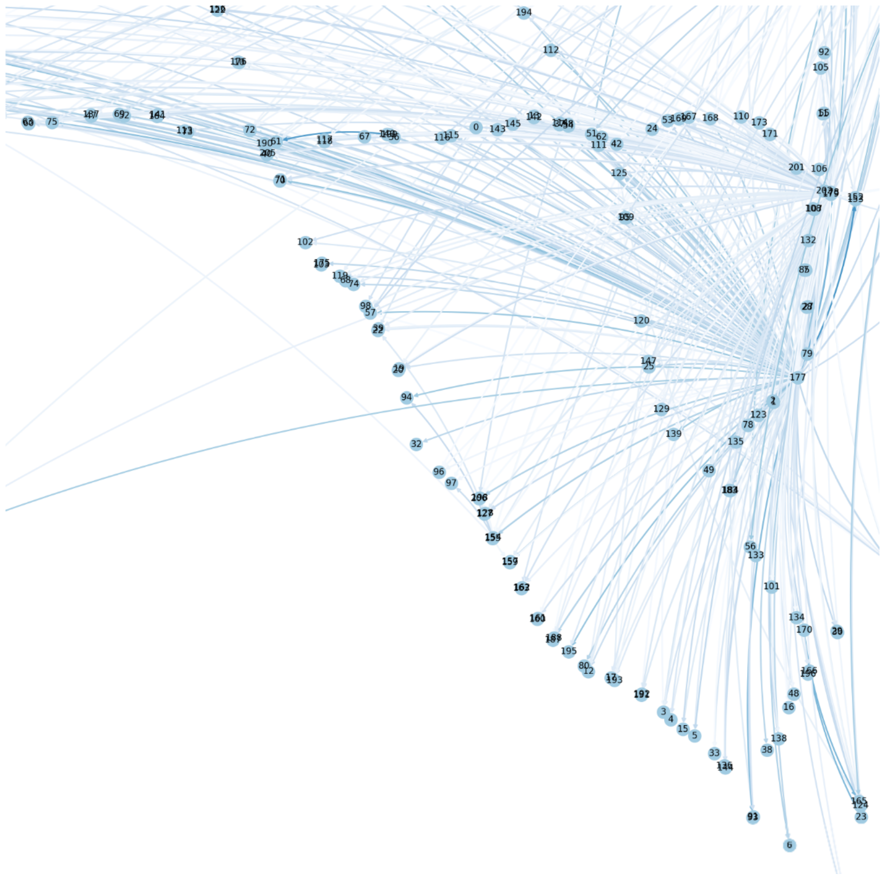


Figure 3.3: Visualization of an inferred causal graph on the METR-LA dataset.

Chapter 4

Bi-level Optimization in Implicit Graph Neural Networks

4.1 Introduction

Graph-structured data is prevalent in real-world scenarios. In order to model and glean insights from such data, graph neural networks (GNNs) were introduced. These networks aim to generate meaningful representations for nodes by simultaneously incorporating information from both the graph topology (edge attributes) and node attributes. Only until recent years, the node-edge co-evolution (NEC) problem setting has attracted growing interest proposed which involves not only node representation but also edge representation learning [23, 28, 29]. This NEC problem is more general than the regular node representation learning task because it can not only be reduced to the later one but also handle more complicated tasks such as deep graph translation [27] and relational inference [39].

Despite the problem setting, a common challenge of GNNs is the lack of ability to capture long-range dependencies. Normally, one layer of any GNN can only capture the dependencies with immediate neighbors from any given node. Stacking multiple layers GNNs can extend the receptive field of GNNs but with a risk of having the so called oversmoothing issue. Besides, stacking too many layers of GNNs also require excessive computational and storage cost since more trainable weights are introduced and computational graph becomes deeper.

Recently, several works have been proposed to overcome the long-range dependency learning challenge in graph data with implicit layers. Implicit GNNs (IGNN) [25] learns node representations by solving an equilibrium equation (DEQ) w.r.t. node attributes. EIGNN solve the problem in the same way but with a different optimization algorithm [47]. However, these works are restricted to the vanilla GNN message passing function. It is non-trivial to make any changes to their message passing function while keeping the characteristic of the implicit layer. Specifically, the NEC problem cannot be solved with existing implicit layers GNNs because the edge representations need to be modeled explicitly.

Motivated by the existing implicit layer GNNs, we propose our NEC-DGT$^{\infty}$ framework with node-edge co-evolution message passing for solving the multi-attributed graph translation problem.

The contribution of this work are summarized as follows:

- To capture long-range dependencies, we propose a novel and generic node-edge co-evolution deep equilibrium model with infinite number of layers of both node-edge and edge-node message passing.
- We derive the well-posedness condition for the node-edge co-evolution message passing to guarantee an unique fixed-point solution can be found.
- We take advantage of the implicit function theorem (IFT) and develop an efficient optimization algorithm for the node-edge co-evolution deep equilibrium model without needing to do layer-by-layer backpropagation.
- We conduct extensive experiments to validate the effectiveness and efficiency of the proposed model. The results show that NEC-DGT$^{\infty}$ can effectively capture long-range dependencies and outperform the state-of-the-art GNN models on both synthetic and

real-world datasets.

4.2 Problem Formulation

4.2.1 Multi-attributed Graph Translation

Table 4.1: Important notations and descriptions

Notations	Descriptions
$G(\mathcal{V}_0, \mathcal{E}_0, E_0, F_0)$	Input graph with node set $\mathcal{V}_0$, edge set $\mathcal{E}_0$, edge attributes tensor E_0 and node attributes matrix F_0
$G(\mathcal{V}', \mathcal{E}', E', F')$	Target graph with node set $\mathcal{V}'$, edge set $\mathcal{E}'$, edge attributes tensor E' and node attributes matrix F'
C	Contextual information vector
$\lvert\mathcal{V}\rvert$	Number of nodes
$\lvert\mathcal{E}\rvert$	Number of edges
B	Node-edge incidence matrix
H_v	node embedding, $H_v \in \mathbb{R}^{p\times\lvert\mathcal{V}\rvert}$
H_e	edge embedding, $H_e \in \mathbb{R}^{q\times\lvert\mathcal{E}\rvert}$
W_{ve}	weight matrix for the node to edge propagation
W_{ev}	weight matrix for the edge to node propagation
$(\ )^{\lvert\cdot\rvert}$	element-wise absolute value of a matrix or vector

A multi-attributed graph is formally defined as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set comprising $|\mathcal{V}|$ nodes, and $\mathcal{E}$ represents the edges. The node attributes tensor, denoted as $F \in \mathbb{R}^{D\times|\mathcal{V}|}$, is characterized by the dimension D of the node attributes. Additionally, the edge attributes tensor, denoted as $E \in \mathbb{R}^{K\times|\mathcal{E}|}$, is defined with K as the dimension of edge attributes.

The task of translating a multi-attributed graph is framed as the learning of a mapping from an input graph, denoted as G_0, to a target graph, denoted as G_T. This mapping is represented as $\mathcal{T} : G_0(\mathcal{V}_0, \mathcal{E}_0, E_0, F_0; C) \rightarrow G_T(\mathcal{V}', \mathcal{E}', E', F')$.

To train the deep neural network for generating the target graph $G(E', F')$ conditioned on the input graph $G(E_0, F_0)$ and contextual information C, we aim to minimize the following loss function:

$$\mathcal{L}_{\mathcal{T}} = \mathcal{L}(\mathcal{T}(G(E_0, F_0), C), G(E', F')) \tag{4.1}$$

4.2.2 IGNN on Ordinary Graphs

Implicit GNNs (IGNN) [25] are graph neural network (GNN) models that learn node representations by solving an equilibrium equation (DEQ) with reference to node attributes. Different from a regular GNN layer, an IGNN layer is defined in terms of satisfying a joint condition, i.e., condition for equilibrium, of the input and output. In other words, an IGNN layer generates the outputs only when the message passing function converges to the unique fixed point. The IGNN on ordinary graphs can be defined as:

$$\begin{aligned} &\hat{Y} = f_{\Theta}(X^t) \\ &X^{t+1} = \phi(W X^t A + b_{\Omega}(F_0)) \\ \text{subject to} \quad & \\ &X^t = X^{t+1} \end{aligned} \tag{4.2}$$

here f_{Θ} is the readout function for the fixed point solution X^t, $X^t \in \mathbb{R}^{p \times |\mathcal{V}|}$ is the node embedding at the t-th iteration of message passing. W is trainable weights, A is the adjacency matrix, b_{Ω} is some affine transformation parameterized by Ω, ϕ is an activation function.

IGNN can be seen as a recurrent neural network on graph with a constraint that a fixed point solution $X^t = X^{t+1}$ must be found. Once the condition $X^t = X^{t+1}$ is satisfied, the output embedding in further iterations will still equal with X^t because all the other variables in the model remain the same. However, the fixed-point solution for the equilibrium equation (4.2) may not exist for arbitrary input and weights. Hence, the concept of well-posedness becomes relevant. This concept was initially introduced in [18] for implicit models and in [25] for standard graph neural networks.

Definition 4.1 (Well-posedness for IGNN)**.** The tuple (W, A) of the weight matrix $W \in \mathbb{R}^{p \times p}$ and the adjacency matrix $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is said to be well-posed for ϕ if for any $B \in \mathbb{R}^{p \times |\mathcal{V}|}$, the solution $X \in \mathbb{R}^{p \times |\mathcal{V}|}$ of the following equation

$$X = \phi(WXA + B) \tag{4.3}$$

exists and is unique.

Existing work proposed efficient algorithms for optimizing the model in Eq. (4.2) while keeping the well-posedness [25, 47]. However, these optimization methods are dependant on the simple message passing function $X^{t+1} = \phi(WX^tA + b_\Omega(F_0))$ which only consists simple node to node propagation. It is non-trivial to change or extend the message passing function.

4.3 Proposed Approach

We now start by describing the proposed NEC-DGT$^\infty$ framework which can infinite-depth node-edge co-evolution. Inspired by the IGNN on ordinary graph, the infinite-depth model for the multi-attributed graph translation problem can be defined as:

$$
\begin{aligned}
&F' = g_{\Theta_e}(H_e^t), E' = f_{\Theta_v}(H_v^t) \\
&H_e^{t+1}, H_v^{t+1} = \mathcal{T}(H_e^t, H_v^t, F_0, E_0) \\
\text{subject to}\quad & \\
&H_e^t = H_e^{t+1} \text{ and } \; H_v^t = H_v^{t+1}
\end{aligned}
\tag{4.4}
$$

here the function $\mathcal{T}$ is applied to both the node and edge embeddings recurrently until it converges to a fixed point. In this way, the message passing procedure is repeated for infinite number of times thus important long range nodes-to-edges, nodes-to-nodes, edges-to-nodes, and edges-to-edges dependencies can be captured. In contrast, in traditional GNNs, only the information of T-hops can be captured when there are T layers of GNNs. The number T in most GNNs is normally small to avoid the the infamous over-smoothing issues in GNNs. We will discuss how our model can avoid over-smoothing issues in later sections.

Node-edge Co-evolution Message Passing

For the message passing function $\mathcal{T}$ in Eq. (4.4) to handle the multi-attributed graph translation problem, it needs to capture important interactions among nodes and edges. Existing work considers four types of interactions in the multi-attributed graph translation problem, including nodes-to-edges interaction, nodes-to-nodes interaction, edges-to-nodes interaction, and edges-to-edges interaction [27]. Due to the nature of recurrent models, we decouple function $\mathcal{T}$ as only two basic paths including the node-to-edge propagation path and the edge-to-node propagation path. The nodes-to-edges interaction and edges-to-nodes interaction can be directly inferred by these two basic propagation paths. Since the message passing procedure is applied iteratively, the nodes-to-nodes interaction can be inferred by stacking one layer of node-to-edge propagation path and one layer of edge-to-node propagation path. Similarly, the edges-to-edges interaction can be inferred by stacking one layer of edge-to-node propagation path and one layer of node-to-edge propagation path. The two propagation paths are defined in the following equations.

$$\begin{aligned} &\text{Node-to-edge:} \quad H_e = \mathcal{T}_1(H_v, B) = \phi(W_{ve} H_v B) \\ &\text{Edge-to-node} \quad H_v = \mathcal{T}_2(H_e, B) = \phi(W_{ev} H_e B^T) \end{aligned} \tag{4.5}$$

Please note that during the node/edge message passing procedures, the node embeddings always remain the same: $H_v \in \mathbb{R}^{p\times|\mathcal{V}|}$, while edge embeddings also remain the same $H_e \in \mathbb{R}^{q\times|\mathcal{E}|}$. To map from the node/edge feature spaces $F \in \mathbb{R}^{D\times|\mathcal{V}|}/E \in \mathbb{R}^{K\times|\mathcal{E}|}$ to the node/edge embedding spaces $H_v \in \mathbb{R}^{p\times|\mathcal{V}|}$ / $H_e \in \mathbb{R}^{q\times|\mathcal{E}|}$, we add a function b_Ω which can capture all four types of interactions for one time. If we explicitly seek node embedding fixed points, the final message passing function is defined as :

$$\begin{aligned} H_v &= \mathcal{T}_v(H_v, B) \\ &= \phi(W_{ev}\phi(W_{ve} H_v B) B^T + b_\Omega(E_0, F_0)) \\ H_e &= \mathcal{T}_e(H_v, B) = \phi(W_{ve} H_v B) \end{aligned} \tag{4.6}$$

Once the fixed-point for node embeddings is found as is shown in Eq. (4.6), the edge embeddings is also automatically found because of the nature of fixed-point functions. Assuming the fixed-point solution for the node embedding is H_v^*, then the edge embedding in the next layer is $\mathcal{T}_e(H_v^*, B)$. As the node embeddings have reached the fixed point, the node embedding in the next layer will still be H_v^*, thus the next of the next edge embedding will still be $\mathcal{T}_e(H_v^*, B)$.

Alternatively, if we seek edge embedding fixed points first, the function b_Ω will be defined to output features in the space of $\mathbb{R}^{q\times|\mathcal{E}|}$. These two solutions are equivalent.

Sufficient Well-posedness Condition for NEC-DGT$^\infty$

For the NEC-DGT$^\infty$ in Eq. (4.6) model to function as we expect it to, i.e., yeild a fixed point solution within finite iterations, it is essential to derive a *distinct* internal state H_v for any input U. To guarantee both the existence and uniqueness of the solution to Eq. (4.6), we introduce the concept of well-posedness for equilibrium equations in the context of multi-attributed graph translation problems.

Definition 4.2 (Well-posedness for NEC-DGT$^\infty$)**.** The tuple (W_{ve}, W_{ev}, B) is considered well-posed for ϕ if, for any $b \in \mathbb{R}^{m\times n}$, the solution $H_v \in \mathbb{R}^{p\times|\mathcal{V}|}$ of the following equation

$$H_v = \phi(W_{ev}\phi(W_{ve}H_vB)B^T + b) \tag{4.7}$$

exists and is unique.

Next, we establish the adequate condition for ensuring the well-posedness, relying on the introduced node-edge co-evolution message passing. As per Definition 4.2, this condition pertains to the interplay among three variables: W_{ve}, W_{ev}, and B.

Theorem 4.3 (PF sufficient condition for well-posedness on Eq. (4.6)**.** *Suppose ϕ is an activation map that is component-wise non-expansive (CONE). In that case, for any such ϕ, the tuple (W_{ve}, W_{ev}, B) is considered well-posed if $\lambda_{pf}((B \otimes W_{ev})^{|\cdot|}(B^T \otimes W_{ve})^{|\cdot|}) < 1$. Furthermore, the solution H_v for equation (4.6) can be iteratively obtained as stated in equation (4.6).*

Proof.

$$H_v = \phi(W_{ev}\phi(W_{ve}H_vB)B^T + b_\Omega(E_0, F_0)) \tag{4.8}$$

vectorize both sides:

$$\begin{aligned}\mathbf{vec}(H_v) &= \phi(\mathbf{vec}(W_{ev}\phi(W_{ve}H_vB)B^T) + \mathbf{vec}(b)) \\ &= \phi((B \otimes W_{ev})\,\mathbf{vec}(\phi(W_{ve}H_v))) + \mathbf{vec}(b))\end{aligned} \tag{4.9}$$

as ϕ is a CONE activation map:

$$\begin{aligned}\mathbf{vec}(H_v^{t+1} - H_v^t)^{|\cdot|} &= (\phi((B \otimes W_{ev})\,\mathbf{vec}(\phi(W_{ve}H_v^tS))) + \mathbf{vec}(B)) \\ &\quad - \phi((B \otimes W_{ev})\,\mathbf{vec}(\phi(W_{ve}H_v^{t-1}B))) + \mathbf{vec}(B)))^{|\cdot|} \\ &\leq ((B \otimes W_{ev})\,\mathbf{vec}(\phi(W_{ve}H_v^tB)) - (B \otimes W_{ev})\,\mathbf{vec}(\phi(W_{ve}H_v^{t-1}B)))^{|\cdot|} \\ &\leq (B \otimes W_{ev})^{|\cdot|}(\mathbf{vec}(\phi(W_{ve}H_v^tB)) - \mathbf{vec}(\phi(W_{ve}H_v^{t-1}B)))^{|\cdot|} \\ &\leq (B \otimes W_{ev})^{|\cdot|}(\mathbf{vec}(W_{ve}H_v^tB) - \mathbf{vec}(W_{ve}H_v^{t-1}B))^{|\cdot|} \\ &= (B \otimes W_{ev})^{|\cdot|}(\mathbf{vec}(W_{ve}(H_v^t - H_v^{t-1})B))^{|\cdot|} \\ &= (B \otimes W_{ev})^{|\cdot|}((B^T \otimes W_{ve})\,\mathbf{vec}((H_v^t - H_v^{t-1})))^{|\cdot|} \\ &\leq (B \otimes W_{ev})^{|\cdot|}(B^T \otimes W_{ve})^{|\cdot|}\,\mathbf{vec}(H_v^t - H_v^{t-1})^{|\cdot|}\end{aligned} \tag{4.10}$$

Eq. (4.6) has a unique solution if $\lambda_{pf}((B \otimes W_{ev})^{|\cdot|}(B^T \otimes W_{ve})^{|\cdot|}) < 1$

$\lambda_{pf}((B \otimes W_{ev})^{|\cdot|}(B^T \otimes W_{ve})^{|\cdot|}) = \lambda_{pf}((BB^T) \otimes (W_{ev}^{|\cdot|}W_{ve}^{|\cdot|})) = \lambda_{pf}(BB^T)\lambda_{pf}(W_{ev}^{|\cdot|}W_{ve}^{|\cdot|}) < 1$

$\square$

While the condition $\lambda_{pf}(BB^T)\lambda_{pf}(W_{ev}^{|\cdot|}W_{ve}^{|\cdot|}) < 1$ in Theorem 4.3 guarantees that the well-posedness can be achieved, calculating the PF eigenvalue is very costly. To avoid costly spectral decomposition process, we enforce the following more strict condition $\|W_{ev}^{|\cdot|}W_{ve}^{|\cdot|}\|_\infty < \lambda_{pf}(BB^T)^{-1}$.

$$\lambda(|W|) = \inf_{S} ||SWS^{-1}||_\infty : S = \mathbf{diag}(S), s > 0 \tag{4.11}$$

$$\lambda_{pf}(W_{ev}^{|\cdot|}W_{ve}^{|\cdot|}) = \inf_{S} ||SW_{ev}^{|\cdot|}W_{ve}^{|\cdot|}S^{-1}||_\infty : S = \mathbf{diag}(S), s > 0 \tag{4.12}$$

If $W_{ev}^{|\cdot|}W_{ve}^{|\cdot|}$ has a distinct PF eigenvalue, the problem (4.12) has a positive optimal scaling factor $s > 0$, which is a PF eigenvector of $W_{ev}^{|\cdot|}W_{ve}^{|\cdot|}$. And we can design the equivalent IGNN with $\|W_{ev}'^{|\cdot|}W_{ve}'^{|\cdot|}\|_\infty < \lambda_{pf}(BB^T)^{-1}$ by rescaling:

$$\tilde{f_\Theta}(\cdot) = f_\Theta(S^{-1}\,\cdot), \quad |W_{ev}'^{|\cdot|}W_{ve}'^{|\cdot|} = SW_{ev}^{|\cdot|}W_{ve}^{|\cdot|}S^{-1}, \quad \tilde{b_\Omega}(\cdot) = Sb_\Omega(\cdot),$$

where $S = \mathbf{diag}(s)$.

Training of NEC-DGT$^\infty$

After deriving the sufficient condition for well-posedness of NEC-DGT$^\infty$, now we can rewrite the loss function as:

$$\begin{aligned}
&\min_{\Theta_v,\Theta_e,W_{ev},W_{ve},\Omega} \mathcal{L}_1(F', g_{\Theta_e}(H_v)) + \mathcal{L}_2(E', f_{\Theta_e}(H_e)) \\
&H_v = \phi(W_{ev}\phi(W_{ve}H_vB)B^T + b_\Omega(E_0, F_0)), \\
&H_e = \phi(W_{ve}H_vB) \qquad\qquad (4.13)\\
&\text{subject to:} \\
&\lambda_{pf}(BB^T)\lambda_{pf}(W_{ev}^{|\cdot|}W_{ve}^{|\cdot|}) < 1
\end{aligned}$$

This optimization problem with constraints can be solved by *projected gradient descent*(PGD), where the gradient is computed using an implicit differentiation scheme. In each optimization step, the PGD approach involves moving in the direction of the negative gradient and subsequently projecting onto the feasible set. The projection itself is also an optimization problem.

Projection

The projection operator can be expressed as the following optimization problem.

$$\begin{aligned} \{W_{ev}^{+}, W_{ve}^{+}\} &= \underset{\lambda_{pf}(W_{ev}^{|\cdot|}W_{ve}^{|\cdot|})<\kappa}{\arg\min} \quad ||W_{ve} - W_{ve}^{+}||_F^2 + ||W_{ev} - W_{ev}^{+}||_F^2 \\ \kappa &= 1/\lambda_{pf}(BB^T) \end{aligned} \tag{4.14}$$

we enforce the stricter condition $||W_{ev}^{|\cdot|}W_{ve}^{|\cdot|}||_\infty < \kappa$

The projection operator now becomes a child optimization problem in Eq. 4.15.

$$\begin{aligned} &\text{minimize} \quad ||X - A||_F^2 + ||Y - B||_F^2 \\ &\text{subject to} \quad ||X^{|\cdot|}Y^{|\cdot|}||_\infty \leq \kappa \end{aligned} \tag{4.15}$$

As the constraint in Eq. (4.15) is about $X^{|\cdot|}$, each dimension in X and A will always be of the same sign. If one dimension in X has the opposite sign with the corresponding dimension in A, the opposite of that dimension will also meet the same constraint but yields a smaller loss. Similarly, each dimension in Y and B will always be of the same sign. For simplicity, we can first minimize $||X^{|\cdot|} - A^{|\cdot|}||_F^2 + ||Y^{|\cdot|} - B^{|\cdot|}||_F^2$. After the optimal $X' = X^{|\cdot|}$ and $Y' = Y^{|\cdot|}$ are found, the actual X and Y can be simply calculated by $X = \mathbf{sign}(A) \odot X'$ and $Y = \mathbf{sign}(B) \odot Y'$. With this trick, the problem in Eq. 4.15 has the following simplified version assuming all positive X, Y, A, and B:

$$\begin{aligned} &\text{minimize} \quad ||X - A||_F^2 + ||Y - B||_F^2 \\ &\text{subject to} \quad XY = C, ||C||_\infty \leq \kappa \end{aligned} \tag{4.16}$$

The optimization problem in Eq. (4.16) can be solved by alternating direction method of multipliers (ADMM) [11]. ADMM solves convex optimization problems by breaking them into multiple simpler ones and solving each of them iteratively. According to ADMM, we first define augmented Lagrangian, for a parameter $\rho > 0$,

$$\begin{aligned} L_\rho(X,Y,C,\lambda) = {} & ||X-A||_F^2 + ||Y-B||_F^2 \\ & + \langle \lambda, XY - C\rangle_F + (\rho/2)||XY-C||_F^2 \end{aligned} \tag{4.17}$$

ADMM consists of the iterations

$$\begin{aligned} & X^{k+1} = \arg\min_X(||X-A||_F^2 + \langle \lambda, XY-C\rangle + (\rho/2)||XY-C||_F^2) \\ & Y^{k+1} = \arg\min_Y(||Y-B||_F^2 + \langle \lambda, XY-C\rangle + (\rho/2)||XY-C||_F^2) \\ & C^{k+1} = \arg\min_{||C||_\infty \le \kappa}(\langle \lambda, XY-C\rangle_F + ||XY-C||_F^2) \\ & \lambda^{k+1} = \lambda^k + \rho(XY-C) \end{aligned} \tag{4.18}$$

For X:

$$\begin{aligned} & \frac{\partial ||X-A||_F^2 + \langle \lambda, XY-C\rangle_F + (\rho/2)||XY-C||_F^2}{\partial X} = 0 \\ & 2(X-A) + \lambda Y^T + \rho(XY-C)Y^T = 0 \\ & 2X - 2A + \lambda Y^T + \rho XYY^T - \rho CY^T = 0 \\ & X(2I + \rho YY^T) = 2A - \lambda Y^T + \rho CY^T \\ & X = (2A - \lambda Y^T + \rho CY^T)(2I + \rho YY^T)^{-1} \end{aligned} \tag{4.19}$$

For Y:

$$\begin{aligned}
&\frac{\partial ||Y-B||_F^2 + \langle \lambda, XY-C\rangle_F + (\rho/2)||XY-C||_F^2}{\partial Y} = 0 \\
&2(Y-B) + X^T\lambda + \rho X^T(XY-C) = 0 \\
&2Y - 2B + X^T\lambda + \rho X^T XY - \rho X^T C = 0 \\
&(2I + \rho X^T X)Y = 2B - X^T\lambda + \rho X^T C \\
&Y = (2I + \rho X^T X)^{-1}(2B - X^T\lambda + \rho X^T C)
\end{aligned} \tag{4.20}$$

For C:

The optimization is separable across the rows of C. Each sub-problem involves projecting onto an $\mathcal{L}_1$-ball, and efficient methods are available for such projections.

Denote optimal primal variables by X^* and Y^*, and the optimal dual variable by λ^*. The primal feasibility is measured by primal residual:

$$r^{k+1} = ||X^{k+1}Y^{k+1} - C^{k+1}||_F \tag{4.21}$$

The dual residual can be defined as:

$$s^{k+1} = \rho X^{k+1}(Y^k - Y^{k+1})Y^{k^T} \tag{4.22}$$

Algorithm 2 summarizes the ADMM optimization iteration.

Gradient Descent based on Implicit Function Theorem

After the projection, we calculate the gradients of loss w.r.t weights in the equilibrium model by utilizing the implicit function theorem.

Algorithm 2: Parameters optimization based on ADMM

```
Input: Weight metrics $A$ and $B$, parameter $\kappa$
Output: Solution $X$ and $Y$
Initialize $\rho = 1$, $C, \lambda,$;
Choose $\varepsilon^{pri} > 0$ , $\varepsilon^{dual} > 0$;
repeat
    Update $X$ by equation (4.19);
    Update $Y$ by equation (4.20);
    Update $C$ by infinity norm;
    Update $\lambda$ by $\lambda^{k+1} = \lambda^k + \rho(XY - C)$;
    Calculate the primal residual by equation (4.21) ;
    Calculate the dual residual by equation (4.22) ;
    if $r > 10s$ then
        $\rho \leftarrow 2\rho$ ;
    else if $10r < s$ then
        $\rho \leftarrow \rho/2$ ;
    else
        $\rho \leftarrow \rho$ ;
    end
until $r < \varepsilon^{pri}, s < \varepsilon^{dual}$;
```

$$H_v = \phi(W_{ev}\phi(W_{ve}H_vB)B^T + b_\Omega(E_0, F_0)) \tag{4.23}$$

From the chain rule, it is easy to obtain $\nabla_{H_v}\mathcal{L}_\infty$ for the internal state X. In addition, we can write the gradient w.r.t scalar $q \in W_{ev}$ as follows:

$$\nabla_q\mathcal{L} = \langle\frac{\partial Z}{\partial q}, \nabla_Z\mathcal{L}\rangle, \tag{4.24}$$

where $Z = W_{ve}H_vB$ assuming fixed H_v.

Unlike H_v that is implicitly defined, Z is a closed evaluation of $Z = W_{ve}H_vB$ assuming H_v doesn't change depending on Z.

To circumvent the need for matrix derivatives, we once again employ the vectorized representation $vec(\cdot)$ for matrices. The vectorization of a matrix $H_v \in \mathbb{B}^{p\times|\mathcal{V}|}$, denoted as $\mathrm{vec}(H_v)$, is achieved by concatenating the columns of H_v into a single column vector with dimensions $p|\mathcal{V}|$. For brevity, we use $\overrightarrow{H_v} := \mathrm{vec}(H_v)$ as a shorthand notation for vectorization.

With vectorization, we have:

$$\overrightarrow{H_v} = \phi((B \otimes W_{ev})\phi((B^T \otimes W_{ve})\overrightarrow{H_v}) + \overrightarrow{B}) \tag{4.25}$$

similarly, now we change the definition of Z

$$\overrightarrow{Z} = ((B \otimes W_{ev})\phi((B^T \otimes W_{ve})\overrightarrow{H_v}) + \overrightarrow{B} \tag{4.26}$$

According to Lemma 4.4, we have

$$\begin{aligned}\frac{\partial \overrightarrow{H_v}}{\partial \overrightarrow{Z}} = &\frac{\partial \phi((B \otimes W_{ev})\phi((B^T \otimes W_{ve})\overrightarrow{H_v}) + \overrightarrow{B})}{\partial \overrightarrow{Z}} \\ &+ \frac{\partial \phi((B \otimes W_{ev})\phi((B^T \otimes W_{ve})\overrightarrow{H_v}) + \overrightarrow{B})}{\partial \overrightarrow{H_v}} \frac{\partial \overrightarrow{H_v}}{\partial \overrightarrow{Z}}\end{aligned} \tag{4.27}$$

here

$$\begin{aligned}&\frac{\partial \phi((B \otimes W_{ev})\phi((B^T \otimes W_{ve})\overrightarrow{H_v}) + \overrightarrow{B})}{\partial \overrightarrow{H_v}} \\ =&\frac{\partial \phi(\overrightarrow{Z})}{\partial \overrightarrow{Z}} \frac{\partial ((B \otimes W_{ev})\phi((B^T \otimes W_{ve})\overrightarrow{H_v}) + \overrightarrow{B})}{\partial \overrightarrow{H_v}} \\ =&\tilde{D_1}(B \otimes W_{ev})^T (B^T \otimes W_{ve})^T \tilde{D_2}\end{aligned} \tag{4.28}$$

where

$$\begin{aligned}\tilde{D}_1 &= \frac{\partial \phi(\overrightarrow{Z})}{\partial \overrightarrow{Z}} \\ \tilde{D}_2 &= \frac{\partial \phi((B^T \otimes W_{ve})\overrightarrow{H_v})}{\partial (B^T \otimes W_{ve})\overrightarrow{H_v}}\end{aligned} \tag{4.29}$$

$$\nabla_{\overrightarrow{Z}}\mathcal{L} = (\frac{\partial \overrightarrow{H_v}}{\partial \overrightarrow{Z}})^T \nabla_{\overrightarrow{H_v}}\mathcal{L} \tag{4.30}$$

Plugging Eq. (4.27) to (4.30), we arrive at the following equilibrium equation

$$\nabla_{\overrightarrow{Z}}\mathcal{L} = \tilde{D}_1 (B \otimes W_{ev})^T (B^T \otimes W_{ve})^T \tilde{D}_2 \nabla_{\overrightarrow{Z}}\mathcal{L} + \tilde{D}_1 \nabla_{\overrightarrow{H_v}}\mathcal{L} \tag{4.31}$$

or in the devectorized form:

$$\nabla_Z \mathcal{L} = D_1 \odot (W_{ev}^T (D_2 \odot (W_{ve}^T \nabla_Z \mathcal{L} B^T))R + \nabla_{H_v}\mathcal{L}) \tag{4.32}$$

here the $\nabla_{H_v}\mathcal{L}$ can be easily obtained through modern autograd frameworks so $\nabla_Z\mathcal{L}$ can be calculated in the fixed-point function in Eq. 4.32.

After $\nabla_Z\mathcal{L}$ is calculated, it is easy to infer the derivative of the loss w.r.t. W_{ev}

$$\nabla_{W_{ev}}\mathcal{L} = \langle \frac{\partial Z}{\partial W_{ev}}, \nabla_Z \mathcal{L} \rangle = \nabla_Z \mathcal{L} R (\phi(W_{ve} H_v B))^T \tag{4.33}$$

For calculating $\nabla_{W_{ve}}\mathcal{L}$, it requires calculating matrix to matrix derivatives so we use the the vectorized representations again.

$$
\begin{aligned}
\nabla_{\overrightarrow{W_{ve}}}\mathcal{L} &= \langle \frac{\partial \overrightarrow{Z}}{\partial \overrightarrow{W_{ve}}}, \nabla_{\overrightarrow{Z}}\mathcal{L}\rangle = \langle \frac{\partial \overrightarrow{Z}}{\partial \overrightarrow{\phi}} \frac{\partial \overrightarrow{\phi}}{\partial \overrightarrow{W_{ve}}}, \nabla_{\overrightarrow{Z}}\mathcal{L}\rangle \\
&= \langle (R \otimes W_{ev}) \frac{\partial \overrightarrow{\phi}}{\partial \overrightarrow{W_{ve}}}, \nabla_{\overrightarrow{Z}}\mathcal{L}\rangle = \langle (R \otimes W_{ev}) \frac{\partial \overrightarrow{\phi}}{\partial \overrightarrow{W_{ve}H_vB}} \frac{\partial \overrightarrow{W_{ve}H_vB}}{\partial \overrightarrow{W_v e}}, \nabla_{\overrightarrow{Z}}\mathcal{L}\rangle \\
&= \nabla_{\overrightarrow{Z}}\mathcal{L}((R \otimes W_{ev})\phi'(\overrightarrow{W_{ve}H_vB})((B^T H_v^T) \otimes I))
\end{aligned}
\tag{4.34}
$$

where I is a $q \times q$ identity matrix.

Thus

$$
\nabla_{W_{ve}}\mathcal{L} = H_v B((B^T \nabla_Z \mathcal{L} W_{ev}) \odot \phi'(W_{ve}H_vB))
\tag{4.35}
$$

Lemma 4.4. *Using Implicit Function Theorem. The function $W \mapsto H$ is defined implicitly by $H = f(H, W)$.*

Then, $H'(W) = \frac{\partial f}{\partial H} H'(W) + \frac{\partial f}{\partial W}$

4.4 Experiments and Results

In this section, we present the evaluation results over the proposed NEC-DGT$^{\infty}$ model. We first introduce the experimental setup, followed by our experimental results and analysis.

Datasets

We conduct experiments on publicly available multi-attributed graph translation datasets including four synthetic datasets and four real-world datasets.

Synthetic datasets: Four synthetic datasets were generated using distinct random graph generators and varying translation rules. In the initial three datasets (labeled Syn-I, Syn-II, and Syn-III), the structures of the input graphs were formed using the Erdős-Rényi (ER) model [19] with different numbers of nodes (20, 40, and 60) and parameters. The target graph's structure is defined as the 2-hop graph of the input graph, signifying that each edge in the target graph corresponds to 2-hop reachability in the input graph. Similarly, the structures of the input graph in the forth dataset (named as Syn-IV) were created by the Barabási-Albert (BA) model [6] with 20 nodes. The structure of the target graph in Syn-IV is the 3-hop graph of the input graph where each edge in the target graph refers to the 3-hop reachability in the graph. The node attributes in the input graphs are the node degrees. The node attributes in the target graphs are calculated as a predefined function of the node attributes in the input graphs. For all four datasets, the edge attribute is a binary variable denoting whether the edge exists or not. 50% data are used for training, 40% are used for validation, and 10% are used for testing.

Malware confinement dataset: Three malware datasets for IoT devices are used for evaluating the performance on malware confinement prediction tasks [16]. In all three datasets, the nodes in the input graph represent IoT devices within the system, with the node attribute being a binary value indicating whether the device is compromised or not. The edge attributes between two nodes are defined as the physical distance between two devices. The target graphs represent the graphs with updated node and edge attributes after after the malware confinement [60].

Molecule reaction dataset: This dataset was collected by Lowe [48] from USPTO granted patents for chemical reaction extraction studies [48] and was used for deep graph translation evaluations [27]. The input graphs represent the molecule of the reactants in organic chemistry, while the target graphs are the products of the chemical reaction. The node attributes

Comparison methods

To evaluate the capability of our proposed method, we compare it with two baseline methods, namely, *NEC-DGT* and *IGNN*. NEC-DGT is a node-edge co-evolution GNN model but with two blocks/layers. IGNN is a DEQ model for node embedding message passing only. In addition, We compare our proposed method with two categories of state-of-the-art methods : 1) graph structure learning (GSL) methods, and 2) node classification/regression methods.

Link attribute prediction / GSL methods. *GT-GAN* is a newly developed generative adversarial network designed for learning graph topology [29]. *GraphRNN* is a deep autoregressive model based on LSTM that can approximate any distribution of graphs with minimal assumptions about their structure. [87], *GraphVAE* is a variational autoencoder-based graph generation method [62].

Node classification/regression methods: *IN* is a versatile graph neural network (GNN) framework designed for learning representations at the node, edge, and graph levels [8]. *DCRNN* is a comprehensive approach that captures both spatial and temporal dependencies through diffusion convolution [44]. *STGCN* constructs ST-Conv blocks with spatial convolution layers and residual connection [89].

Evaluation metrics

The graph translation problem is a multi-objective machine learning problem. While node attributes and edge attributes are used as input at the same time, the node prediction and edge prediction are evaluated separately. For different datasets, the target attributes, for both nodes and edges, can be binary values or continuous values. We use MSE, R2, Pearson's r and Spearman's r as metrics. For binary values, we use accuracy as the metric.

Performance

1) Metric-based evaluation for synthetic datasets: All the results are shown in table 4.2. It can be seen that the proposed model outperform all comparison methods on both the

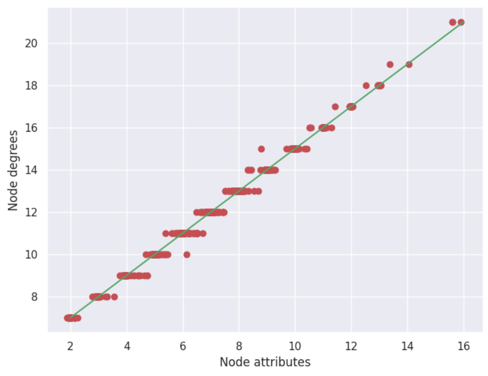
node and edge attributes prediction. Specifically, in terms of node attribute prediction, the methods that consider both node and edge messaging perform (NR-DGT, NEC-DGT, NEC-DGT$^{\infty}$) significantly better than the methods that only do node-to-node message passing (IN, DCRNN, STGCN). With the help of the DEQ architecture, NEC-DGT$^{\infty}$ performs even better than NEC-DGT. It is worth to note that on the Syn-IV dataset for BA graphs, the MSE has been decreased one order of magnitude compared with the second best performed method. In terms of edge attribute prediction, the proposed NEC-DGT$^{\infty}$ also win by a large margin. Notably, for Syn-III, the highest accuracy that the other methods can achieve was only 65.8% while NEC-DGT$^{\infty}$ reached a 95.6% accuracy with a 45% improvement. Also notably, the edge prediction accuracy on Syn-IV has been improved to almost 100%.

2) Evaluation of the learned translation mapping for synthetic data:

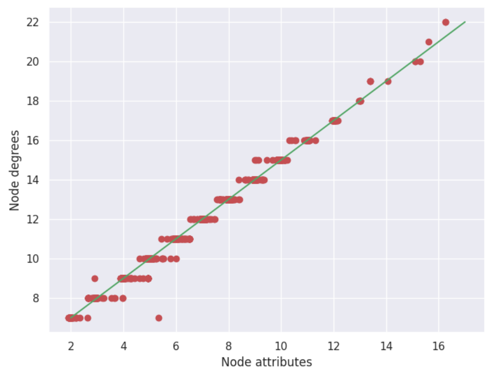
To illustrate whether the inherent mapping mechanism for both node and edge attributes in the graph translation problem is learnt correctly by NEC-DET$^{\infty}$, we visualize the ground-truth mapping and plot the learnt distribution by NEC-DET$^{\infty}$ for all the four synthetic datasets. The ground-truth mapping is drawn according to the pre-defined functions of generating the datasets as described in section 4.4. Figure 4.1 shows groundtruth line in green and predicted values in red dots. As is shown in Figure 4.1, the predicted values are located closely around the groundtruth.

3) Metric-based Evaluation for realworld datasets

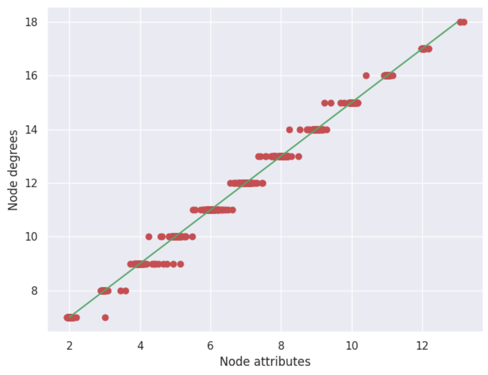
Performance metrics for the malware detection task is shown in Table 4.3. The edge attributes are continuous values and thus are evaluated by E-MSE, E-R2, and E-P. The node attributes are evaluated by E-Acc. NEC-DGT$^{\infty}$ achieves the overall best performance. In terms of edge attributes prediction, NEC-DGT$^{\infty}$ outperform all the comparison methods. For node attributes prediction, NEC-DGT$^{\infty}$ performs the best on the first two datasets.



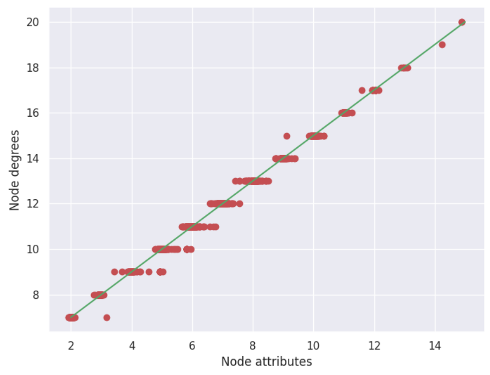
(a)



(b)



(c)



(d)

Figure 4.1: Visualizations between predicted node attributes and the ground truth relationship for samples from four synthetic graphs. (a) Syn-I (b) Syn-II (c) Syn-III (d) Syn-IV

On Malware III, NEC-DGT$^{\infty}$ gets a slightly lower accuracy than GT-GAN but can have a much lower E-MSE. In summary, the node-edge message passing methods (NEC-DGT and NEC-DGT$^{\infty}$) can handle node and edge prediction at the same time better than the rest methods. By introducing the infinite-depth node-edge message passing, NEC-DGT$^{\infty}$ consistently performs better than its finite layer counterpart (i.e., NEC-DGT)

4.5 Discussion

This study focuses on a new problem, end-to-end attributed graph transformation. To achieve this, we propose a novel NEC$^{\infty}$ method consisting of a graph deep equilibrium model which translates an initial graph to a target graph with different node and edge attributes. To jointly tackle complicated node-edge dynamics, the infinite-depth node-edge co-evolution message passing is proposed. We also proposed an efficient implicit theorem-based optimization algorithm to avoid heavy computation and memory overhead. To the best of our knowledge, NEC$^{\infty}$ is the first work of its kind that is capable of incorporating both node and edge dynamics within an equilibrium architecture. Extensive experiments have been conducted on both synthetic and real-world datasets. Experiment results have shown that our NEC$^{\infty}$ can approximate the underlying ground-truth translation rules, even those with iterative graph-wide operations, and it significantly outperforms existing methods and baselines.

Table 4.2: Assessment of Generated Target Graphs for Synthetic Dataset (N for node attributes, E for edge attributes, P for Pearson correlation, SP for Spearman correlation, and Acc for accuracy)

Dataset	**Method**	**N-MSE**	**N-R2**	**N-P**	**N-Sp**	**Method**	**E-Acc**
Syn-I	IN	5.97	0.06	0.48	0.44	GraphRNN	0.6212
	DCRNN	51.36	0.12	0.44	0.45	GraphVAE	0.6591
	STGCN	15.44	0.19	0.42	0.56	GT-GAN	0.7039
	NR-DGT	2.13	0.87	0.90	0.89	NR-DGT	0.7017
	NEC-DGT	1.98	0.76	0.93	0.91	NEC-DGT	0.7129
	NEC-DGT$^\infty$	**1.018**	**0.931**	**0.965**	**0.959**	NEC-DGT$^\infty$	**0.9445**
Syn-II	IN	1.36	0.85	0.77	0.87	GraphRNN	0.5621
	DCRNN	71.07	0.11	0.39	0.37	GraphVAE	0.4639
	STGCN	33.11	0.21	0.15	0.15	GT-GAN	0.7005
	NR-DGT	1.43	0.91	0.94	0.97	NR-DGT	0.7016
	NEC-DGT	1.91	0.93	0.97	0.97	NEC-DGT	0.7203
	NEC-DGT$^\infty$	**1.203**	**0.963**	**0.982**	**0.983**	NEC-DGT$^\infty$	**0.9691**
Syn-III	IN	35.46	0.31	0.59	0.56	GraphRNN	0.4528
	DCRNN	263.23	0.09	0.41	0.39	GraphVAE	0.3702
	STGCN	43.34	0.22	0.48	0.47	GT-GAN	0.5770
	NR-DGT	5.90	0.90	0.94	0.92	NR-DGT	0.6259
	NEC-DGT	4.56	0.93	0.97	0.96	NEC-DGT	0.6588
	NEC-DGT$^\infty$	**3.322**	**0.953**	**0.976**	**0.974**	NEC-DGT$^\infty$	**0.9559**
Syn-IV	IN	4.63	0.10	0.53	0.51	GraphRNN	0.5172
	DCRNN	63.03	0.12	0.22	0.16	GraphVAE	0.3001
	STGCN	6.52	0.08	0.11	0.10	GT-GAN	0.8052
	NR-DGT	4.49	0.12	0.55	0.54	NR-DGT	0.6704
	NEC-DGT	1.86	0.73	0.93	0.89	NEC-DGT	0.8437
	NEC-DGT$^\infty$	**0.183**	**0.972**	**0.986**	**0.985**	NEC-DGT$^\infty$	**0.9988**

Table 4.3: Assessment of Generated Target Graphs for Malware Dataset (N for node attributes, E for edge attributes, P for Pearson correlation, SP for Spearman correlation, and Acc for accuracy).

Malware-I						
Method	E-Acc	E-MSE	E-R2	E-P	Method	N-Acc
GraphRNN	0.6107	1831.43	0.52	0.00	IN	0.8786
GraphVAE	0.5064	2453.61	0.00	0.04	DCRNN	0.8786
GT-GAN	0.6300	1718.02	0.42	0.11	STGCN	0.9232
NR-DGT	0.9107	668.57	0.82	0.91	NR-DGT	0.9108
NEC-DGT	0.9218	239.79	0.78	0.91	NEC-DGT	0.9295
NEC-DGT$^{\infty}$	**0.9384**	**195.72**	**0.78**	**0.92**	NEC-DGT$^{\infty}$	**0.9347**

Malware-II						
Method	E-Acc	E-MSE	E-R2	E-P	Method	N-Acc
GraphRNN	0.7054	1950.46	0.44	0.29	IN	0.8828
GraphVAE	0.6060	2410.57	0.73	0.16	DCRNN	0.8790
GT-GAN	0.9033	462.73	0.13	0.81	STGCN	0.9330
NR-DGT	0.9117	448.48	0.68	0.83	NR-DGT	0.8853
NEC-DGT	0.9380	244.40	0.81	0.91	NEC-DGT	0.9340
NEC-DGT$^{\infty}$	**0.9417**	**233.66**	**0.81**	**0.92**	NEC-DGT$^{\infty}$	**0.9487**

Malware-III						
Method	E-Acc	E-MSE	E-R2	E-P	Method	N-Acc
GraphRNN	0.8397	1775.58	0.16	0.23	IN	0.8738
GraphVAE	0.8119	2109.64	0.39	0.32	DCRNN	0.8738
GT-GAN	0.9453	550.30	0.63	0.80	STGCN	0.9375
NR-DGT	0.9543	341.10	0.76	0.88	NR-DGT	0.8773
NEC-DGT	0.9604	273.67	0.81	0.90	NEC-DGT	0.9002
NEC-DGT$^{\infty}$	**0.9783**	**231.99**	**0.83**	**0.92**	NEC-DGT$^{\infty}$	**0.92875**

Chapter 5

Bi-level Optimization in Neural Architecture Search for Echo-State Networks

5.1 Introduction

Graphs are commonly used as universal representations of real-world things, including social networks, brain functional connections, and molecular topology. Real-world networks generally exhibit patterns in their dynamics, which may be classed as "dynamics **on** graphs" and "dynamics **of** graphs". The former stresses the time-evolving patterns of the entities' activity, which can be proven explicitly through the observable node attributes, whereas the latter emphasizes the underlying change in the topological structure of the network. Both forms of dynamics appear in real-world graphs, and it is tremendously advantageous to understand their linkage and transformation. In social networks, for instance, it is crucial to investigate how the node-level behaviors might affect time-evolving connectivities [21]. In neuroscience, it is essential to examine how the co-activation of many neurons increases their physical nerve connections [50]. In recent years, a substantial amount of work and knowledge has been devoted to "dynamics graphs," a mix of "dynamics on graphs" and "dynamics of graphs." Dynamic graph embedding methods, for instance, compute dynamic node embedding by aggregating messages from nodes' neighborhoods, which requires the input of both node signals (i.e., dynamics on graphs) and graph topology (i.e., dynamics of graphs) [53, 59, 65, 96]. In practice, however, it is typically quite difficult to directly measure all the edges to immediately perceive the entire graph. Instead, it is considerably more frequent and economical to deduce the underlying network structure from the node signals. Therefore, we propose the task that transfers "dynamics on graphs" to "dynamics of graphs" (in short, "**on-to-of**" task) to map the two individual spaces.

Existing on-to-of efforts can be divided into two distinct categories. The first class of approaches discretizes continuous node signals before implementing message passing on a fully connected [53, 85], resulting in a severe loss of information. The second category encodes full time series directly into their embeddings and then calculates the correlation between these embeddings using standard metrics such as cosine similarity [23, 32, 39, 68]. However, such metrics are usually not flexible and powerful, and thus cannot adapt to more complicated on-to-of mappings.

This study focuses on the on-to-of task, which cannot be effectively addressed by existing solutions due to the following challenges: **(1) Difficulty in jointly extracting features from node dynamics while learning the dynamic relationships in a graph.** The challenge necessitates that transformation patterns be considered in both time and graph dimensions. Moreover, these two dimensions are not independent, necessitating the need for a framework that can facilitate the combined evolution of node and edge dynamics. **(2) Absence of an effective and scalable framework for graph dynamics encoding over a continuous long time duration with a high sampling rate.** The inference of dynamic graph topology necessitates fine-grained, long-term knowledge on graph dynamics. Existing efforts for dynamic network embedding and representation learning are unable to efficiently manage extended time series of node attribute data. **(3) Dilemma between**

learnability and efficiency of models. Modeling the complex mapping between node and edge dynamics demands models with a high capacity for learning. Compared to optimizing a model fitted to specific data, optimizing a highly flexible, highly learnable model is typically time-consuming.

To address the above challenges, we present a novel framework based on echo-state network (ESN) and neural architecture search (NAS). Specifically, a deep echo-state network is proposed to efficiently encode the continuous time series of node attributes into dynamic edge embeddings. The architecture of the ESN is automatically tuned by NAS in a self-supervised manner. The application of the ESN makes the framework extremely efficient and scalable when dealing with continuous node signals. The NAS module enables the framework to be adaptive to varying data with minimum priors and ensures good results. The contributions of this study are as follows:

(1) Design a novel generic framework for mapping between "dynamics on graphs" and "dynamics of graphs". A generic framework is proposed for learning the mapping from the continuous-time node attributes to relational graph topology in an end-to-end manner.

(2) Propose an adaptive deep echo-state network for encoding node dynamics toward dynamic edge embedding. The proposed deep echo-state network-based encoder overcomes the inefficiency of BPTT for long time series data.

(3) Design the first neural architecture search method for echo-state networks. The search space of the echo-state network is defined as the binary connectivity of the reservoirs in an echo-state network. The optimization of the architecture is formulated as a bi-level optimization problem which can be solved by gradient-based algorithms.

(4) Conduct extensive experiments for performance and efficiency evaluations. The proposed method was evaluated on both synthetic and real-world application data. The results demonstrate that the proposed approach runs significantly more efficiently and exhibits better performance than the baseline methods.

5.2 Problem Formulation

In this section, basic concepts and problem definitions are introduced.

Definition 5.2.1 (Dynamics <u>ON</u> Graphs)**.** In a graph with V nodes, the dynamics on graph are defined as the multivariate time series sensed continuously on all the nodes denoted as $S = \{S^{(1)}, S^{(2)}, \cdots, S^{(V)}\}$, where $S^{(i)}$ is the node signal for node i. $S^{(i)}$ can be defined as a sequence of time series segments $S^{(i)} = \{S^{(i)}_1, S^{(i)}_2, \cdots, S^{(i)}_m\}$, where each segment is multivariate time series. For convenience, we also denote $S = \{S_1, S_2, \cdots, S_m\}$ for m time series segments, where $S_k = \{S^{(1)}_k, S^{(2)}_k, ..., S^{(V)}_k\}$. $S^{(i)}_k$ can be further defined as discrete time series with length l: $S^{(i)}_k = \{s^{(i)}_{k,1}, s^{(i)}_{k,2}, ..., s^{(i)}_{k,l}\}$

Definition 5.2.2 (Dynamics <u>OF</u> Graphs)**.** The dynamics of graphs $G = \{G_1, G_2, \cdots, G_m\}$ is an ordered sequence of distinct weighted graphs on the set of nodes V, with each snapshot graph $G_k(V, E_k, \mathcal{A}_k)$, where each edge set at the k-th time series segment is denoted as $E_k \subseteq V \times V$. Each graph G_k corresponds to a weighted adjacency matrix $\mathcal{A}_k \in \mathbb{R}^{V \times V}$. $\mathcal{A}_{k,i,j} > 0$ denotes there exists an edge $e_{k,i,j} \in E_k$ from node $v_i \in V$ to $v_j \in V$, and $\mathcal{A}_{k,i,j} = 0$, otherwise. The ordered sequence of adjacency matrix can also represent the dynamics of a graph $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_m\}$

Definition 5.2.3 (The **<u>on-to-of</u>** task)**.** Given the dynamics on graphs $S = \{S_1, S_2, \cdots, S_m\}$ for m time segments on all nodes, we assume that there exist ground truth underlying

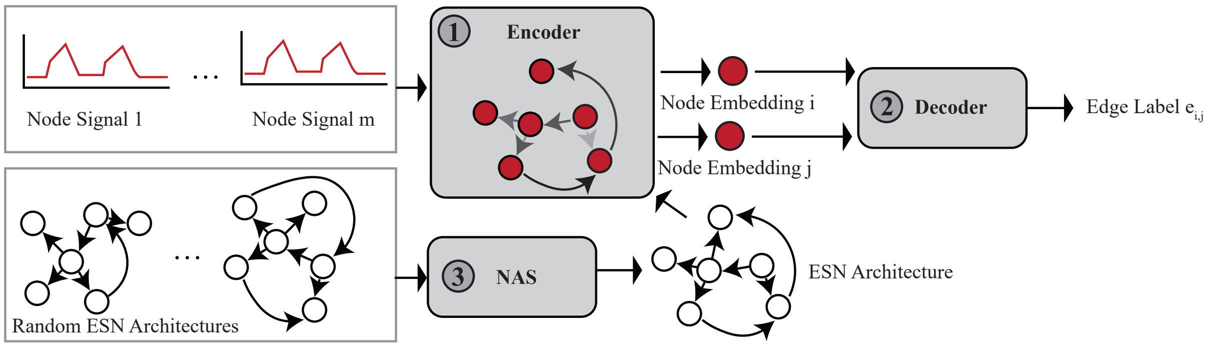


Figure 5.1: Adaptive deep echo-state framework for the on-to-of task. 1: Deep-echo-state graph dynamics encoder; 2: Dynamic graph topology decoder; 3: Deep-echo-state architecture optimization

dynamics of graphs $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_m\}$ that represent the time-evolving relationship among these nodes. The on-to-of task is to infer a function to map between $\mathcal{S}$ and $\mathcal{A}$: $\mathcal{F} : \mathcal{S} \to \mathcal{A}$.

The on-to-of task is different from most of the dynamic graph studies because only the continuous node signals are used as input and the evolving graph topology is the output. In most dynamic graph studies, the dynamic graph topology must be used as well in a graph neural network [53, 85]. The on-to-of task is also different from the more commonly studied GSL problem [97] where the evolving graph topology (i.e., dynamics of graphs) is optimized towards optimizing a downstream task. For instance, in [23], the downstream task is the auto-regression with a rational behind is: accurately recovered interactions (i.e., graph topology) permit accurate node trajectory forecasting (i.e., the downstream task). In the on-to-of task, though, recovering the dynamic graph topology is the task itself.

5.3 Proposed Approach

To solve the on-to-of task and address the challenges, we propose a new adaptive deep echo-state framework for graph dynamics transformation in this section.

5.3.1 Adaptive Deep Echo-state Framework

The adaptive deep echo-state framework (AD-ESN) mainly includes three modules as demonstrated in Fig. 5.1. To solve the challenge of lacking scalability, we extend echo state network (ESNs) and propose a deep ESN-based graph dynamics encoder (module 1 in Fig. 5.1) to replace mainstream recurrent models because ESNs are much more efficient and scalable than RNNs. The details of using ESNs as time series data encoders are explained in Section 5.3.2. In order to improve the performance and make the model adaptive, we propose a novel idea of using a neural architecture search technique on ESNs (module 3 in Fig. 5.1). This solution not only solves the challenge of lacking model assumptions for the on-to-of task but also remedies the shortage of vanilla ESNs that the performance is poor. The details are discussed in Section 5.3.4. ESN and NAS together enable us to learn meaningful embeddings of arbitrary node signals in a graph. We use an attention-based dynamic graph topology decoder (module 2 in Fig. 5.1) for mapping the node embeddings to edge labels as detailed in Section 5.3.3. While NAS is a powerful technique, it can also be extremely slow, which does not match our original intention of proposing an efficient on-to-of task solver. We solve this issue by proposing a surrogate loss and using a gradient-based optimization algorithm which is much faster to solve as detailed in Section 5.3.5.

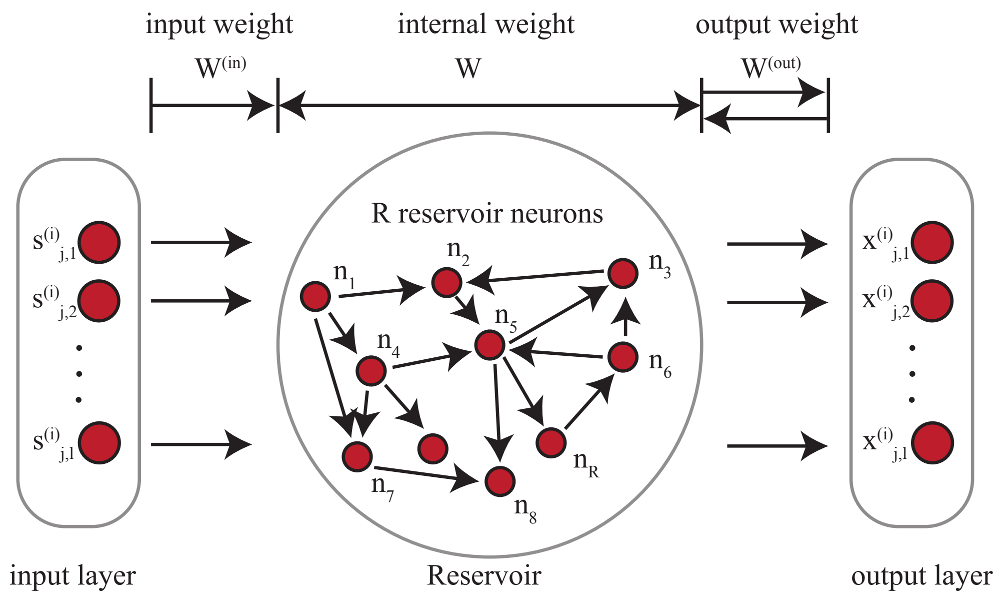


Figure 5.2: Echo State Network. In our work, the architecture of the reservoir is optimized with NAS.

5.3.2 Deep-echo-state Graph Dynamics Encoder

We propose an adaptive deep ESN-based encoder for learning the representation of the dynamics of graphs. The input dynamics, which can be considered as time series, are non-linearly embedded into a higher dimensional feature space, where the original problem is more likely to be solved linearly according to Cover's Theorem [14]. With proper settings, the dependencies of our ESN-based graph dynamics encoder on the initial conditions are progressively lost, and the state of the network asymptotically depends only on the driving input signal.

Our ESN-based encoder can be considered as a recurrent neural network where all of the weights are randomized and untrained. As shown in Fig. 5.2, there are input weight, internal weight, and output weight. On the left side of the figure, each timestamp in the input time series is considered as an input node. $W^{(in)} \in \mathbb{R}^R$ is the weight that maps each input node to the internal reservoir neurons. $W_i^{(in)} = 0$ means no connection from the input to the i-th reservoir neuron. Every time a new timestamp $S_k^{(i)}$ is fed in, the reservoir neurons are affected by not only the input data but also the current state of all the neurons in the ESN. Similarly, the output weight $W^{(out)}$ connects the state of the reservoir neurons to the outputs. The dynamics of the ESN can be represented as $r_t = \sigma(W r_{t-1} + W^{(in)} s_{k,t}^{(i)})$, where σ is a nonlinear activation function, r_t is the current state of the ESN reservoir at time t, $W \in \mathbb{R}^{R \times R}$ is the weight among the R reservoir neurons (shown in Fig. 5.2). The non-zero values in W can be represented as directed edges in the ESN. Different from vanilla ESNs, the non-zero weights of $W^{(in)}$ and W are formulated as the search space of deep ESN architectures as described in Section 5.3.4, and are optimized with NAS as detailed in Section 5.3.5. The $W^{(out)}$ in Fig. 5.2 generates hidden representations of the input such that we can use them for the following on-to-of task solver.

We denote $\mathcal{R}$ as the ESN's reservoir, and $x_k^{(i)}$ as the representation of the k-th time series segment of node i. $x_k^{(i)}$ can be calculated as the final hidden representation in the ESN that can be computed recursively as shown in Eq (5.1):

$$\begin{aligned} x_k^{(i)} &= W^{(out)} \mathcal{R}(S_k^{(i)}) \\ &= W^{(out)} \sigma((W r_{k,l-1}^{(i)} + W^{(in)} s_{k,l}^{(i)})) \\ &= W^{(out)} \sigma(W \sigma(W r_{k,l-2}^{(i)} + W^{(in)} s_{k,l-1}^{(i)}) + W^{(in)} s_{k,l}^{(i)}) \end{aligned} \tag{5.1}$$

In conclusion, the ESN does not *learn* the representation of the input time series. It is directly applied to the sequential input data and maps the input into a high-dimensional space. While ESN is efficient, tuning its initial state is highly dependant on domain experts' experiences. It is highly desired if ESNs can be automatically tuned before it's trained on the labeled data.

5.3.3 Dynamic Graph Topology Decoder

The proposed attention-based dynamic graph topology decoder aims to infer time-evolving edge features or connectivities, ensuring that the on-to-of task inference is scalable and general without bias. Within the k-th time series segment, the dynamics on graphs now are represented as embeddings $X_k = \{x_k^{(1)}, x_k^{(2)}, \cdots, x_k^{(V)}\}$ on V nodes. We define the attention coefficients between node i and node j as the edge label (dynamics of graphs) we try to infer: $e_{k,i,j} = a(x_k^{(i)}, x_k^{(j)})$, where a is the attentional mechanism with shared weights. As there is already a shared learnable linear transformation layer W_{out} in the ESN before getting the embeddings, an external linear transformation is omitted before using the attention mechanism.

To facilitate the comparability of coefficients across various nodes in the dynamic graph, the attention coefficients are normalized using the softmax function. For example, the attention mechanism a can be a single head GAT [70] (denoted as g) parameterized with $W^{(G)}$ defined on a fully connected graph. Now we can express the on-to-of task as:

$$\begin{aligned}\mathcal{A}_{k,i,j} &= F_{(W^{(out)},W^{(G)})}(S_k^{(i)}, S_k^{(j)}) \\ &= \frac{exp(\sigma(g(W^{(out)}\mathcal{R}(S_k^{(i)})||W^{(out)}\mathcal{R}(S_k^{(j)}))))}{\sum_{n\in V} exp(\sigma(g(W^{(out)}\mathcal{R}(S_k^{(i)})||W^{(out)}\mathcal{R}(S_k^{(n)}))))}\end{aligned}$$

5.3.4 AD-ESN Search Spaces

As no NAS studies have been proposed for ESNs, we propose *ESN architecture search*, the first attempt of using NAS on ESNs for graph dynamics learning. The goal is to automatically learn the ESN architecture so that it will achieve a good performance in downstream tasks. The following is the search space that we define for ESNs.

- The connectivity from the input to ESN's reservoir neurons $A^{(in)} \in \mathbb{R}^{d \times R}$ where $A^{(in)}_{i,j} = \{0, 1\}$. $A^{(in)}_{i,j} = [W^{(in)}_{i,j} \neq 0]$ [1]

- The connectivity between ESN's reservoir neurons $A \in \mathbb{R}^{R \times R}$ where $A_{i,j} = \{0, 1\}$. $A_{i,j} = [W_{i,j} \neq 0]$

For simplicity we denote $A^{(R)} = [A^{(in)}, A]$ as the ESN's architecture. The choice of activation functions and the number of reservoir nodes are also hyperparameters in the search space but can be set according to general rules (will be discussed in Section 5.4).

5.3.5 Deep-echo-state Architecture Optimization

A rigorous optimization loss function for optimizing AD-ESN can be expressed as follows:

$$\begin{aligned}
&A^{(R)} = \arg\min_{A} \mathcal{L}(W_1^*, W_2^*, A) \\
&\mathcal{L}(W_1, W_2, A) = \sum_{0<k<m} \sum_{i \neq j} |F_{(W_1, W_2, A)}(S_k^{(i)}, S_k^{(j)}) - \mathcal{A}_{k,i,j}| \\
&\text{where } [W_1^*, W_2^*] = \arg\min_{[W_1, W_2]} \mathcal{L}(W_1, W_2, A)
\end{aligned} \tag{5.2}$$

here $F_{(W_1, W_2, A)}$ represents the graph topology decoder with a fixed ESN architecture A.

However, the loss in Eq. (5.2) is extremely expensive to solve as it requires doing the bi-level optimization on the original on-to-of task. One common practice in NAS studies is to propose a surrogate loss that is much easier to solve. Inspired by self-supervised learning, instead of optimizing the ESN for the empirical loss in Eq. (5.2), we decouple the ESN-based encoder and the graph topology decoder, then define a surrogate loss function for the prediction

[1]Please note that A represents the graph structure of ESNs, while $\mathcal{A}$ represents the graph structure of the target graphs (dynamics of graphs)

performance of the ESN-based encoder. Given the time series data $S' = \{s'_1, s'_2, \cdots, s'_m\}$ sampled from the whole data S, the NAS problem with the surrogate loss is defined in Eq. (5.3).

$$\begin{aligned}
A^{(R)} &= \arg\min_{A} \mathcal{L}_s(W^{(pred*)}, A) \\
\mathcal{L}_s(W, A) &= \sum_{0<t<m-1} |\mathcal{R}'_{W,A}(S'_{0:t}) - s'_{t+1})| \\
W^{(pred*)} &= \arg\min_{W} \mathcal{L}_s(W, A^{(R)})
\end{aligned} \tag{5.3}$$

here the previous $W^{(out)}$ is replaced with $W^{(pred)}$. Different from $W^{(out)}$, $W^{(pred)}$ transforms the ESN reservoir's internal states into a representation that has the same dimensionality as the time series data. $\mathcal{R}'_{W,A}$ represents the reservoir with W as output weight and A as architecture. The rationale behind this surrogate loss is similar with NRI [39]: good architectures permit accurate forecasting.

To solve the problem in Eq. (5.3) efficiently, we sample ESN graph connections with Gumbel-Softmax by using the reparameterization trick. Now the problem has been transformed from the optimization in the discrete space into the optimization in a continuous space and can be optimized with stochastic gradient descent. To avoid overfitting and improve ESN's generalization capability, the time series S' in Eq. (5.3) are split into the training set and validation set.The bi-level optimization can be summarized as in Alg. 5.1.

Algorithm 5.1 (htb).

```
1: initialize $A^{(R)}$ in continuous space
2: while Early stopping criterion is not met do
3:     for $e$ in epoch do
4:         for minibatch in training and validation data do
5:             //Sample discrete ESN according to $A^{(R)}$
6:             $A^{(R)} \sim Gumbel(0, 1)$
7:             Initialize $W^{(in)}$ and $W$ according to $A^{(R)}$
```

```
8:           // Update the ESN's predict weights on the training set
9:           $W = W - \eta_W \nabla_W \mathcal{L}_{tr}(W, A^{(R)})$
10:          // Update the ESN's topology parameters $A^{(R)}$ on the validation set
11:          $A^{(R)} = A^{(R)} - \eta_{A^{(R)}} \nabla_{A^{(R)}} \mathcal{L}_{val}(W, A^{(R)})$
12:        end for
13:    end for
14: end while
```

For each batch of data, the ESN architecture is sampled according to the continuous $A^{(R)}$ (lines 6). The ESN's prediction weight $W^{(pred)}$ can be optimized by regular backpropagation (lines 9). The architecture parameter $A^{(R)}$ of the Echo State Network (ESN) is optimized using a straightforward heuristic, where the optimization of the validation loss assumes that the current $W^{(pred)}$ is equivalent to $W^{(pred')}$ (lines 11). After the optimization is finished, to form the discrete topology in the ESN, we retain the edge labels (exist or not) by applying a threshold λ. More details of the NAS process can be found in Appendix B.

5.3.6 Complexity Analysis

We discuss the time complexity of the AD-ESN framework by comparing it with the traditional LSTM and BPTT-based method. The time complexity of our ESN-based encoder module is $\mathcal{O}(Rl)$, where R is the number of internal reservoir neurons and l is the length of the time series. Before our method, the standard way of handling time series data with recurrent neural networks is BPTT (backpropagation through time), which is $\mathcal{O}(R^2 l)$ [77]. The time complexity of the graph topology decoder can be expressed as $\mathcal{O}(V^2)$. This is due to the necessary pairwise computation of nodes' hidden representations.

There is also a NAS module in our framework that takes time, while a vanilla RNN model does not require it. However, the NAS is meant to automate the fine-turning process that was originally performed by humans, which normally takes a longer time. Furthermore, our proposed NAS process is independent of the supervised learning process and only runs on small sampled unlabelled data.

5.4 Experiments

Datasets

The proposed AD-ESN framework and baseline methods are tested on five datasets including two synthetic and three real-world datasets.

- **Syn-Coupled** [39]: Physical simulated dataset for phase-coupled oscillators. In this dataset, each node is an oscillator that is coupled with its neighbors according to a dynamic graph.
- **Syn-Chaotic:** Physical simulated chaotic time series data. In this dataset, each node is defined as a chaotic time series. The ground truth dynamic graphs are defined as real-time correlation graphs.
- **Brain** [63] :Real-world brain fMRI data. The dynamic graphs represent the functional connectivity between brain regions. The node signals represent the BOLD time series in each of the brain regions.
- **Social** [20]: Real-world social media dataset. The node signals are forum users' activities. The dynamic graphs are users' accumulated transition graphs.
- **Protein** [2]: Real-world protein folding data. The node signals contain the amino acids' 3-dimensional dynamic coordinates. The dynamic graphs are the protein's connectivity

during the folding process.

For Syn-Chaotic, Forum, and Brain datasets, the edges have dynamic weights, such that the "dynamics of graphs" are represented as affinity matrices. We evaluate the results with average MAE and RMSE. For Syn-Coupled and Protein datasets, the "dynamics of graphs" are represented as adjacency matrices. The results are evaluated with accuracy (Acc) following [39]. We also report the recall (sensitivity) rate as it is important for the model to have fewer false negatives, i.e., discover the edge if it exists. More detailed data descriptions can be found in Appendix A.

Experiment Settings

All the models are trained with the ADAM optimization algorithm. For each of the datasets, the first 80% of the data are used as the training set, and the rest 20% are used as the testing set. The architecture of the ESN is optimized on 10% of the training data with a gradient-based NAS algorithm. For AD-ESN, we apply *tanh* as the nonlinear function, as it is the most common activation function. As a general rule of thumb, for an input of size d, to remember τ time points in the past, the number of ESN nodes should be at least $d \times \tau$ [49]. Now, the search space is only limited by $A^{(R)}$, which controls the connections on the ESN nodes. This search space is discrete (connected or not at each edge) and has finite 2^{dR+RR} states. The randomly generated ESN weights are normalized to meet a standard called Echo State Property (ESP). More implementation details can be found in Appendix B.

To show AD-ESN's strengths in terms of adaptability and scalability, we compare it with two variants: *LSTM-Att* utilizes LSTM as encoder and GAT as decoder. *ESN* adopts vanilla ESN as encoder and GAT as decoder. Additionally, we compare AD-ESN with two recent SOTA approaches for graph inference: *NRI* [39] uncovers the relation graph by learning a variational auto-encoder (VAE). *dNRI* [23] is similar with NRI but encodes the temporal dependence with LSTM. At last, two simple baselines are also being compared: *Pre-step* simply determines the relations between nodes as the previous status in the last segment. *Siamese* [52] encodes the time series with LSTM and decode the graph dynamics with a siamese feed-forward neural network.

Performance and Adaptability Analysis

	Dynamic Edge Binary Classification						Dynamic Edge Weight Estimation					
Datasets	Syn-Coupled-1		Syn-Coupled-2		Protein		Syn-Chaotic		Forum		Brain	
Metrics	Acc	Recall	Acc	Recall	Acc	Recall	RMSE	MAE	RMSE	MAE	RMSE	MAE
Pre-step	51.2	72.5	47.7	70.3	64.5	68.7	0.036	0.033	0.33	0.067	0.8	0.505
LSTM-Att	51.4	62.5	51.7	49.9	53.3	45.3	0.036	0.033	0.26	0.047	-	-
Siamese	66.2	14.6	53.2	11.3	50.0	47.3	0.039	0.035	0.26	0.052	-	-
ESN	50.0	48.6	50.4	48.3	50.0	42.4	0.031	0.03	0.49	0.091	0.93	0.562
NRI	**94.6**	**91.5**	49.9	33.5	49.9	48.2	**0.029**	**0.026**	0.43	0.075	-	-
dNRI	94.4	91.4	56	33.7	56.3	62.6	**0.029**	**0.026**	0.49	0.088	-	-
AD-ESN	92.2	90.9	**74.4**	**83.6**	**67.8**	**71.4**	**0.029**	**0.026**	**0.26**	**0.05**	**0.67**	**0.442**

Table 5.1: On-to-Of Task Performance Comparison. "-" denotes that the model is not trainable on our hardware.

Table 5.1 summarizes the results of all the models on all the datasets. We observe that AD-ESN achieves overall the best performance on all five datasets with different data scales and underlying priors, which indicates the exceptional adaptability of AD-ESN over the baselines. Some of the baseline models can perform well on a restricted set of tasks but fall short on others, which means they are much more sensitive to datasets. For simulated coupled oscillator data, we first test all the methods on a sampled dataset with the same initialization circumstances as NRI (Syn-Coupled-1) [39], then another dataset with a different configuration (Syn-Coupled-2). It can be seen that NRI and dNRI perform better than AD-ESN on Syn-Coupled-1, but fails miserably on Syn-Coupled-2 when the hyperparameters are not fine-tuned. Our proposed AD-ESN architecture, on the other hand, performs slightly worse than NRI and dNRI on Syn-Coupled-1 but significantly better than all the rest comparison approaches. Appendix A contains the details of the data generation method. On the remaining datasets, our proposed AD-ESN consistently outperforms the other methods. Only two of the ESN-based algorithms are scalable to be trained on the Brain dataset since the time-series of nodes are excessively long.

The predicted graphs using AD-ESN are compared to the ground truth graphs from the Syn-Chaotic dataset in Fig. 5.3. The adaptability of AD-ESN is enabled by the proposed deep-echo-state graph dynamics encoder which is automatically altered with NAS in a self-supervised manner.

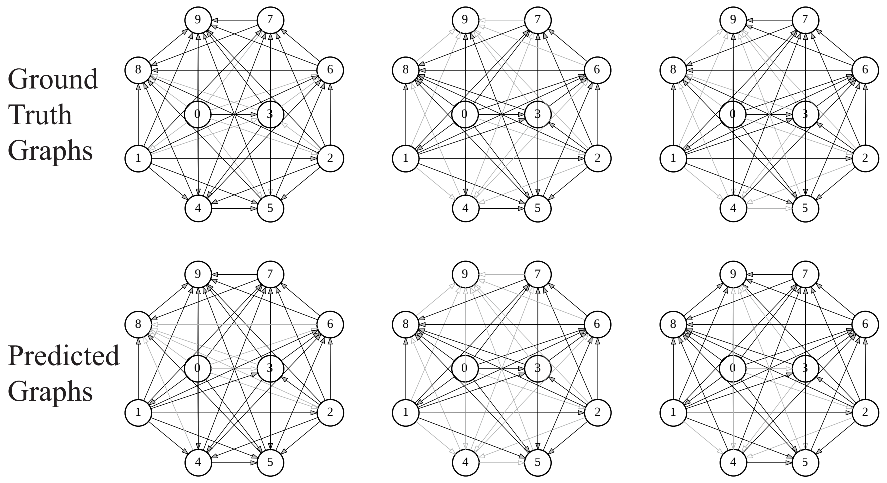


Figure 5.3: Visualization of graphs. Darkness of the edges reflects their weights.

Scalability Analysis

Model	L	D	Node #	BS	GPU Memory
LSTM-Att	500	100	50	128	8893 MB
AD-ESN	500	100	50	128	6425 MB
LSTM-Att	1000	10	50	128	6581 MB
AD-ESN	1000	10	50	128	1647 MB
LSTM-Att	5000	1	50	128	N/A
LSTM-Att	5000	1	50	↓ 32	7464 MB
AD-ESN	5000	1	50	128	**2227 MB**

Table 5.2: GPU Usage Test Results.

L: Time Series Length, D: Input dimension, BS: Batch size.
N/A means the GPU memory is not enough for this setting.

The excellent scalability of AD-ESN stems from the fact that the weights of ESNs are ran-

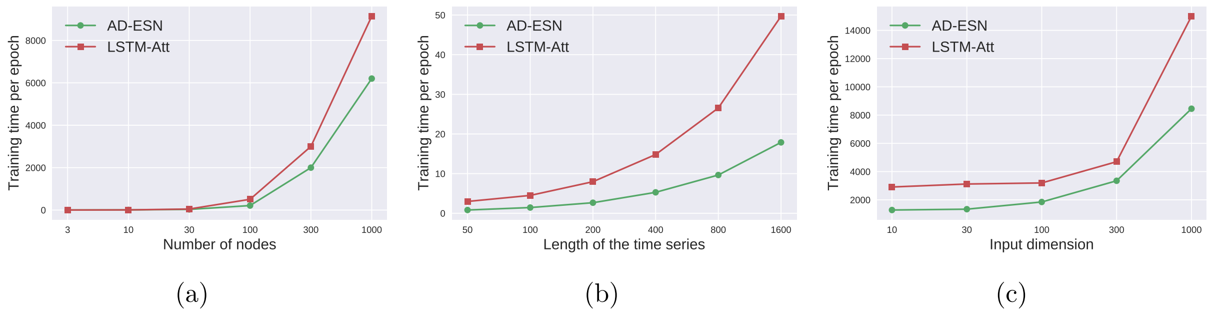


Figure 5.4: Scalibility Analysis. The ESN encoder makes the framework much more efficient and scalable.

domized and untrained, which is different from all existing neural network modules. To better illustrate this feature, we investigate AD-ESN's scalability by comparing it to its counterpart using LSTM as encoder, i.e., LSTM-Att.

Table 5.2 shows a comparison of LSTM-Att and AD-ESN models' GPU RAM usage on synthetic data. The hidden dimensions of both models are the same. The LSTM-Att model utilizes more RAM than the AD-ESN model when used to process long time series. The advantage increases when the time series data become longer, because RNN-based models (e.g., LSTM, GRU) require memory to store the gradients for each timestamp backpropagation, whereas ESNs do not.

A training time comparison between models employing the ESN-based encoder and its LSTM-based counterpart (LSTM-Att) is shown in Fig. 5.4. The training cost of the NAS process is negligible compared with the actual training time due to the efficient gradient-based bi-level optimization and the surrogate loss. While the ESN-based encoder is only one component of the whole framework, the training time of the whole AD-ESN framework is clearly shorter in all scenarios, especially when the length of time series increases, which coincides with the time complexity analysis in Section 5.3.6.

5.5 Discussion

This research has focused on solving the generic "dynamics on graphs" to the "dynamics of graphs" translation tasks without knowing what type of mapping (e.g., causal, correlation, physical interaction) was employed. To do so, we have proposed a novel ESN-based framework with NAS that can automatically tune its architecture based on the input continuous node signal data in a self-supervised manner. To the best of knowledge, this is the first work that combines ESN and NAS. This combination enables the framework to achieve a compelling trade-off between the efficiency and neural architecture flexibility.

Experiment results attest that our AD-ESN framework can successfully uncover the underlying on-to-of mappings on different types of data. The employment of ESN and NAS has been proven to be surprisingly effective and makes the framework highly versatile and scalable.

This study opens a new window for more scalable dynamic graph modeling and new applications with adaptive ESNs. There are many possible future directions to explore, such as online "dynamics on graphs" to the "dynamics of graphs" translation and unsupervised "dynamics on graphs" to the "dynamics of graphs" translation.

Chapter 6

Conclusion

This thesis is about formulating and solving bi-level optimization problems in deep learning. We studied both methodologies and their applications from the lens of bi-level optimization. In In Chapter 2,

In Chapter 3,

modeling intelligence that can learn to represent and reason about the world. We studied both representation and reasoning from the lens of graph neural networks. In Part I, we introduced theoretical frameworks for characterizing the representation power and developed maximally powerful GNNs. In Part II, we answered what reasoning a neural network can learn by analyzing how the interplay of network structure and task structure affects the generalization. In Part III, we studied how neural networks extrapolate, and showed implications for how neural models can possibly learn the reasoning outside the training distribution. Then in Part IV, we completed the picture of the theoretical landscape by analyzing and improving the optimization of GNNs. From a broader viewpoint, we have presented several theoretical limits of building artificial general intelligence in terms of representation power and generalization, but also have complemented those with promising directions to explore in the future. This concludes the thesis.

Bibliography

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] Namrata Anand and Po-Ssu Huang. Generative modeling for protein structures. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7505–7516, 2018.

[3] L Bai, L Yao, C Li, X Wang, and C Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In *34th Conference on Neural Information Processing Systems*, 2020.

[4] Lei Bai, Lina Yao, Salil S. Kanhere, Xianzhi Wang, and Quan Z. Sheng. Stg2seq: Spatial-temporal graph to sequence model for multi-step passenger demand forecasting. In *IJCAI-19*, pages 1981–1987, 7 2019.

[5] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *International Conference on Learning Representations*, 2017.

[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[7] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[8] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Inter-

action networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

[9] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Nan Rosemary Ke, Sebastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryxWIgBFPS.

[10] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, pages 610–619. PMLR, 2018.

[11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[12] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129(3):638–655, 2021.

[13] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. {DARTS}-: Robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KLH36ELmwIB.

[14] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334, 1965.

[15] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[16] Sai Manoj Pudukotai Dinakarrao, Hossein Sayadi, Hosein Mohammadi Makrani, Cameron Nowzari, Setareh Rafatirad, and Houman Homayoun. Lightweight node-level malware detection and network-level malware confinement in iot networks. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 776–781. IEEE, 2019.

[17] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. *Advances in Neural Information Processing Systems*, 33, 2020.

[18] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Tsai. Implicit deep learning. *SIAM Journal on Mathematics of Data Science*, 3(3):930–958, 2021.

[19] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[20] Y. Gao, Lingfei Wu, Houman Homayoun, and Liang Zhao. Dyngraph2seq: Dynamic-graph-to-sequence interpretable learning for health stage prediction in online health forums. *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1042–1047, 2019.

[21] Yuyang Gao, Tanmoy Chowdhury, Lingfei Wu, and Liang Zhao. Modeling health stage development of patients with dynamic attributed graphs in online health communities. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[22] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In *AAAI'19*, 2019.

[23] Colin Graber and Alexander Schwing. Dynamic neural relational inference for forecasting trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 1018–1019, 2020.

[24] CWJ Granger. Testing for causality: a personal viewpoint. In *Essays in econometrics: collected papers of Clive WJ Granger*, pages 48–70. 2001.

[25] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. *Advances in Neural Information Processing Systems*, 33: 11984–11995, 2020.

[26] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929, 2019.

[27] Xiaojie Guo, Liang Zhao, Cameron Nowzari, Setareh Rafatirad, Houman Homayoun, and Sai Manoj Pudukotai Dinakarrao. Deep multi-attributed graph translation with node-edge co-evolution. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 250–259. IEEE, 2019.

[28] Xiaojie Guo, Liang Zhao, Zhao Qin, Lingfei Wu, Amarda Shehu, and Yanfang Ye. Interpretable deep graph generation with node-edge co-disentanglement. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1697–1707, 2020.

[29] Xiaojie Guo, Lingfei Wu, and Liang Zhao. Deep graph translation. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[30] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11993–12002, 2020.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] Jaroslav Hlinka, David Hartman, Martin Vejmelka, Jakob Runge, Norbert Marwan, Jürgen Kurths, and Milan Paluš. Reliability of inference of directed climate networks using conditional mutual information. *Entropy*, 15(6):2023–2045, 2013.

[33] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[34] Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks. In *Advances in Neural Information Processing Systems*, pages 1886–1896, 2019.

[35] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[36] Svante Janson, Andrzej Rucinski, and Tomasz Luczak. *Random graphs*. John Wiley & Sons, 2011.

[37] Yufan Jiang, Chi Hu, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. Improved differentiable architecture search for language modeling and named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3576–3581, 2019.

[38] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 2020–2029, 2018.

[39] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697. PMLR, 2018.

[40] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[41] Kun Kuang, Peng Cui, Bo Li, Meng Jiang, Shiqiang Yang, and Fei Wang. Treatment effect estimation with data-driven variable decomposition. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[42] Dawid Laszuk. Python implementation of kuramoto systems, 2017. URL https://github.com/laszukdawid/Dynamical-systems.

[43] Holden Lee, Rong Ge, Tengyu Ma, Andrej Risteski, and Sanjeev Arora. On the ability of neural nets to express distributions. In *Conference on Learning Theory*, pages 1271–1296. PMLR, 2017.

[44] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR '18)*, 2018.

[45] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

[46] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[47] Juncheng Liu, Kenji Kawaguchi, Bryan Hooi, Yiwei Wang, and Xiaokui Xiao. Eignn: Efficient infinite-depth graph neural networks. *Advances in Neural Information Processing Systems*, 34:18762–18773, 2021.

[48] Daniel Mark Lowe. *Extraction of chemical structures and reactions from the literature*. PhD thesis, University of Cambridge, 2012.

[49] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer, 2012.

[50] Guixiang Ma, Nesreen K Ahmed, Theodore L Willke, Dipanjan Sengupta, Michael W Cole, Nicholas B Turk-Browne, and Philip S Yu. Deep graph similarity learning for brain data analysis. In *CIKM 2019*, pages 2743–2751, 2019.

[51] Michael C Mackey and Leon Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.

[52] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *thirtieth AAAI conference on artificial intelligence*, 2016.

[53] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370, 2020.

[54] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[55] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

[56] Robin Ru, Pedro Esperança, and Fabio Maria Carlucci. Neural architecture generator optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12057–12069. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/8c53d30ad023ce50140181f713059ddf-Paper.pdf.

[57] Bidisha Samanta, Abir De, Niloy Ganguly, and Manuel Gomez-Rodriguez. Designing random graph models using variational autoencoders with applications to chemical design. *arXiv preprint arXiv:1802.05283*, 2018.

[58] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[59] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th international conference on web search and data mining*, pages 519–527, 2020.

[60] Hossein Sayadi, Hosein Mohammadi Makrani, Sai Manoj Pudukotai Dinakarrao, Tinoosh Mohsenin, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 728–733. IEEE, 2019.

[61] Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3839–3848, 2018.

[62] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer, 2018.

[63] Stephen M Smith, Christian F Beckmann, Jesper Andersson, Edward J Auerbach, Janine Bijsterbosch, Gwenaëlle Douaud, Eugene Duff, David A Feinberg, Ludovica Griffanti, Michael P Harms, et al. Resting-state fmri in the human connectome project. *Neuroimage*, 80:144–168, 2013.

[64] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[65] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. Learning to represent the evolution of dynamic graphs with recurrent models. In *Companion proceedings of the 2019 world wide web conference*, pages 301–307, 2019.

[66] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[67] Mingxing Tan and Quoc V Le. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*, 2021.

[68] Liubov Tupikina, Nora Molkenthin, Cristóbal López, Emilio Hernández-García, Norbert Marwan, and Jürgen Kurths. Correlation networks from flows. the case of forced and time-dependent advection-diffusion dynamics. *PloS One*, 11(4):e0153703, 2016.

[69] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL `http://arxiv.org/abs/1609.03499`.

[70] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

[71] Bochao Wang, Hang Xu, Jiajin Zhang, Chen Chen, Xiaozhi Fang, Ning Kang, Lanqing Hong, Wei Zhang, Yong Li, Zhicheng Liu, Zhenguo Li, Wenzhi Liu, and Tong Zhang. Vega: Towards an end-to-end configurable automl pipeline, 2020.

[72] Yaoming Wang, Yuchen Liu, Wenrui Dai, Chenglin Li, Junni Zou, and Hongkai Xiong. Learning latent architectural distribution in differentiable neural architecture search via variational information maximization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12312–12321, 2021.

[73] Yuandong Wang, Hongzhi Yin, Hongxu Chen, Tianyu Wo, Jie Xu, and Kai Zheng. Origin-destination matrix prediction via graph convolution: a new perspective of passenger demand modeling. In *KDD 2019*, pages 1227–1235, 2019.

[74] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[75] Chen Wei, Chuang Niu, Yiping Tang, and Ji min Liang. Npenas: Neural predictor guided evolution for neural architecture search. *ArXiv*, abs/2003.12857, 2020.

[76] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[77] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 433, 1995.

[78] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 1907–1913, United States of America, 2019. Association for the Advancement of Artificial Intelligence (AAAI). doi: 10.24963/ijcai.2019/264. URL https://ijcai19.org/. International Joint Conference on Artificial Intelligence 2019, IJCAI-19 ; Conference date: 10-08-2019 Through 16-08-2019.

[79] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1284–1293, 2019.

[80] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[81] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai

Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2019.

[82] Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? *Advances in Neural Information Processing Systems*, 33, 2020.

[83] Shen Yan, Kaiqiang Song, Fei Liu, and Mi Zhang. Cate: Computation-aware neural architecture encoding with transformers. In *ICML*, 2021.

[84] Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HygrdpVKvr.

[85] Yu Yang, Jiannong Cao, Milos Stojmenovic, Senzhang Wang, Yiran Cheng, Chun Lum, and Zhetao Li. Time-capturing dynamic graph embedding for temporal linkage evolution. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[86] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

[87] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018.

[88] Jiaxuan You, Jure Leskovec, Kaiming He, and Saining Xie. Graph structure of neural networks. In *International Conference on Machine Learning*, pages 10881–10891. PMLR, 2020.

[89] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[90] Miao Zhang, Steven Su, Shirui Pan, Xiaojun Chang, Ehsan Abbasnejad, and Reza Haffari. idarts: Differentiable architecture search with stochastic implicit gradients. In *ICML*, 2021.

[91] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. *Advances in neural information processing systems*, 32, 2019.

[92] Qi Zhang, Jianlong Chang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. Spatio-temporal graph structure learning for traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1177–1185, 2020.

[93] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1234–1241, 2020.

[94] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018.

[95] Daquan Zhou, Qibin Hou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. Rethinking bottleneck structure for efficient mobile network design. *ECCV, August*, 2, 2020.

[96] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[97] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. A survey on graph structure learning: Progress and opportunities. *arXiv e-prints*, pages arXiv–2103, 2021.

[98] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Appendices

Appendix A

Supplementary Material for Generative Neural Architecture Search

A.1 A. Theoretical Analysis and Proofs

A.1.1 Expressivity of the Neural DAG generator

Our neural DAG generator is a neural network that transforms random vectors with Gaussian prior into random graphs with an unknown distribution. To theoretically describe the expressivity of neural DAG generator, a key ingredient is the classic Barron's Theorem [7], which gives a Fourier criterion for approximation ability of a function by a neural network with 1 hidden layer. According to Barron's theorem, if a certain quantity involving the Fourier transform is small, the function can be approximated by a neural network with one hidden layer and a small number of nodes. The similar studies regarding to distributions, not univariate functions, was less studied until just recently [43].

Lemma 2.2 provides expressively guarantee for neural networks on transforming a Gaussian distribution to another arbitrary distribution, whereas Proposition A.1 requires proving the neural network's ability on transforming a Gaussian vector to neural DAG's distribution. A

Barron function in Lemma 2.2 is a function that can be approximated by a neural net with 1 hidden layer. Lemma 2.2 has been proved in [43] which partly explains the expressive power of MLP as generative models.

Proposition A.1. *Our neural DAG network generator can approximate the common graph generators including ER, BA, and WS used in existing work such as RandWire and NAGO with asymptotical zero error.*

Proof. The distribution of ER, BA, and WS graphs are pre-defined with limited parameters:

- The Erdős–Rényi model has two parameters n and p where n is the number of nodes. When n is fixed, the degree distribution of an ER graph is $P(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$, where p is the only free parameter.
- The Barabási–Albert model has two parameters n and m where n is the number of nodes. When n is fixed, the degree distribution of a BA graph is $P(k) = k^{-3}$, where m is the only free parameter.
- The Watts–Strogatz model has three parameters n, k, and β, where n is the number of nodes. When n is fixed, the degree distribution of a WS graph is $P(k) \approx \sum_{n=0}^{f(k,K)} \binom{K/2}{n} (1-\beta)^n \beta^{K/2-n} \frac{(\beta K/2)^{k-K/2-n}}{(k-K/2-n)!} e^{-\beta K/2}$, where only two free parameters p and k can be adjusted.

For each of the above graph generators, the distributions is a Barron function to its free parameter(s). According to Lemma 1, the neural DAG generator (an MLP) can approximate ER, BA, and WS graph generators. □

Remark A.2. (Expressivity issues with methods not using a generator). DARTS-like frameworks optimize the local architectures directly in the form of adjacency matrix $E \in \mathbb{R}^{\frac{n(n-1)}{2} \times N_{op}}$ such that they cannot handle the isomorphic graph issue.

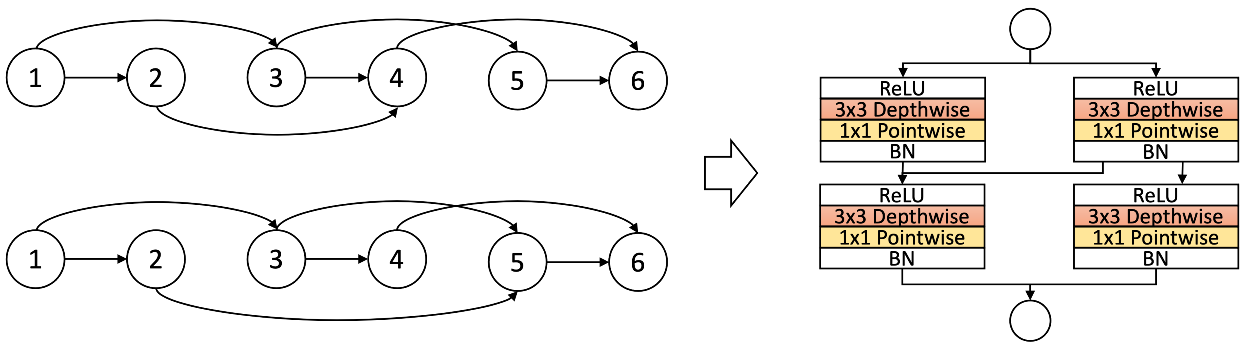


Figure A.1: The left two DAGs represent the same architecture on the right side.

Fig. A.1 shows an example of isomorphic graphs of the same neural network architecture. In DARTS-like frameworks, the goal of the optimization is on the local edge $E^{i \to j}$. However, as shown in the figure, the optimal option $E^{i \to j}$ on the edge can vary depending on different isomorphic graphs of the same architecture.

A.1.2 Continuity of the gNAS Framework

Proposition A.3. *Our neural DAG generator is Lipschitz continuous.*

Proof. The Lipschitz constant of an affine function $f : y = Mx + b$ where $M \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ is the largest singular value of its associated matrix M.

For any MLP with 1-Lipschitz activation functions (e.g., ReLU, Tanh, Sigmoid), the Lipschitz constant becomes

$$L = \prod_{k=1}^{K} ||M_k||_2 \tag{A.1}$$

where K is the number of layers in the MLP.

As the neural DAG generator is an MLP, we have proved that it is Lipschitz continuous. The upper bound of the Lipschitz constant is $\prod_{k=1}^{K} ||M_k||_2$ □

Proposition A.4. *Given the training and validation data, the loss function of the neural DAG-based neural network is Lipschitz continuous with respect to the graph edges.*

Proof. Given a neural DAG-based neural network with two nodes, where the number of potential operations is 2, the output of the continuous-relaxed neural network defined in Section 4.3 is

$$f(\alpha) = \alpha f^1_{w_1}(x) + (1-\alpha) f^2_{w_2} \tag{A.2}$$

where $f^i_{w_i}$ is the function of the i-th operation parameterized with w_i. α is the weight/probability of the first operation.

We have

$$\begin{aligned} & |f(\alpha_1) - f(\alpha_2)| \\ =& |\alpha_1 f^1_{w_1}(x) + (1-\alpha_1) f^2_{w_2} - \alpha_2 f^1_{w_1}(x) - (1-\alpha_2) f^2_{w_2}| \\ =& |(\alpha_1 - \alpha_2)[f^1_{w_1}(x) - f^2_{w_2}(x)]| \end{aligned} \tag{A.3}$$

As all the potential operations are bounded, $[f^1_{w_1}(x) - f^2_{w_2}(x)]$ is also bounded. We denote $|f(\alpha_1) - f(\alpha_2)| \leq L_{out}|\alpha_1 - \alpha_2|$. It means that given the training and validation data, the *output* of the graph-based neural network is Lipschitz continuous with respect to the graph edges.

Given a graph with more than two nodes, it can be proven that the output of the graph-based neural network is still Lipschitz continuous with respect to the graph edges by recursively merging two edges as one edge in Eq. A.2.

Under the same assumption that there are only two nodes and two potential operations, now we try to prove there exists a L that satisfies:

$$|[f(\alpha_1) - y^*]^2 - [f(\alpha_2) - y^*]^2| \leq L|\alpha_1 - \alpha_2| \tag{A.4}$$

where y^* is the ground truth labels.

$$|[f(\alpha_1) - f(\alpha_2)][f(\alpha_1) + f(\alpha_2) - 2y^*]| \leq L|\alpha_1 - \alpha_2| \tag{A.5}$$

As $|f(\alpha_1) - f(\alpha_2)| \leq L_{out}|\alpha_1 - \alpha_2|$ and $|\alpha_1 - \alpha_2| \in [0, 1]$, L exists.

Similarly, this conclusion can be extended to the situation where the graph has more than two nodes. □

A.2 B. Experiment Settings

Our implementation of the gNAS framework is based on Vega AutoML pipeline[71] with PyTorch backend[1]. We use the existing codes for the NAS and evaluation pipelines but rewrite the codes for the search space definition and search algorithm. Our codes include but not limited to:

- Search Algorithm
  - vega/core/search_algs/ps_differential.py

  Some codes for first order differentiable NAS was available in the Vega framework. We implement second order differentiation for gNAS by referring to codes in [46]. The

[1]https://github.com/huawei-noah/vega

weights in the neural DAG generator can be optimized with the differentiable NAS algorithm.

- Search Space
  - zeus/modules/operators/cell.py
  - zeus/networks/super_network.py
  - zeus/networks/pytorch/customs/utils/logical_graph.py
  - zeus/networks/pytorch/customs/utils/layers.py

  We implement the basic operations and RNAG/HNAG networks in above codes.

A.3 C. Related Work

Graph Structures of Neural Network

[88] proposes a universal graph-based representation of neural network architectures and found that the performance of neural network architectures is highly correlated with certain graph characteristics. [17] utilizes graph convolutional network (GCN) to estimate architecture performance, but the search space is not all the possible graphs. These works indicate the complexity of the graph topology and its close relationship with NAS. RandWire was proposed to generate random graphs with prescribed network models (e.g., ER, BA, and WS models), and then transform the graphs into neural network architectures [79]. Later, [56] proposed a framework named NAGO with a hierarchical graph, where in each layer, the graph is a random graph, similar to RandWire. Different from RandWire in that NAS is not utilized, NAGO uses Bayesian optimization (BO) for adjusting the node and edge distribution on each layer. However, methods like RandWire and NAGO are not able to

model the inner graph topology in details. Compared with RandWire and NAGO, we take a step further by optimizing a neural network as the network generator, which is more flexible than the random graph models they use.

Graph Generative Models

Graph generative models have been researched for many years in the domain of mathematics, physics, and statistics to model the behavior of networks. Traditional random network models (e.g., Erdős–Rényi (ER) model [19], Barabási–Albert (BA) model [1], and Watts–Strogatz (WS) model [74]) usually fit well towards the properties that have been covered by the predefined principles, but not on those have not been covered [54]. By leveraging the universal approximation properties of neural networks [80], deep graph generative models can learn and fit arbitrarily complex graph distribution from massive data, without prior knowledge as in the traditional network models. Various neural networks have been proposed for graph representation learning including those based on multi-layer perception [62], graph convolution [40], and graph recurrent nets [87], all of which can be learned by scenarios such as variational auto-encoders (VAE) [57], GAN [10], and maximal likelihood [87]. In recent years, few deep graph generative models have started to be applied to NAS such as in D-VAE [91], arch2vec [82], and CATE [83]. These methods require to learn from the pre-defined network typologies that are generated beforehand. Instead, in our proposed gNAS framework, the stochastic neural DAG generator can be trained in end-to-end fashion and explore the neural architecture without the need for predefined architecture and can avoid the bias and restrictions from prior knowledge.

NAS Optimization Algorithm

NAS by its nature requires high computational demand because the feedback of its generalization ability is expensive to calculate when the architecture is changed. NASNet uses a reinforcement learning-based method to spend 2000 GPU days to obtain the best architecture for CIFAR-10 and ImageNet [98]. Similarly, AmoebaNet-A spends 3150 GPU days using an evolutionary algorithm (EA) [55]. [38] combine BO and EA, but the search space is limited by the initial pool of pre-defined architectures. DARTS-like frameworks (e.g., DARTS [46], P-DARTS [12], PC-DARTS [81], I-DARTS [37], DARTS$-$ [13], MileNAS [30]) speeds up the NAS process by transforming the discrete neural architecture space into a continuously differentiable form, and further uses gradient-based optimization techniques to search the neural architecture. However, the DARTS-like frameworks only search for deterministic local structures (i.e, the operation on a discrete edge in a DAG) and cannot capture the global wiring patterns. In our problem formulation, we assume that a graph generator g can encode global characteristics a class of neural networks, and we search for the optimal weights of the graph generator such that it can generate good architectures consistently.

Appendix B

Supplementary Material for Neural Architecture Search for Echo-State Networks

B.1 A. Datasets

Syn-Chaotic

This synthetic dataset is for verifying if the proposed model can learn a pre-defined mapping function without any prior knowledge. We assume each node generates a chaotic time series that exhibits complex nonlinear behavior but with a deterministic property. The synthetic chaos time series can be generated from the Mackey-Glass function [51].

$$\frac{dx}{dt} = \beta \frac{x_\tau}{1 + {x_\tau}^n} - \gamma x, \quad \gamma, \beta, n > 0, \tag{B.1}$$

where x_τ represents the value of the variable x at time $(t - \tau)$. Depending on the values of the parameters, this equation displays a range of periodic and chaotic dynamics. For a commonly used setup where $\beta = 0.2, n = 10, \gamma = 0.1$, the system has a chaotic attractor for $\tau > 16.8$.

In the experiment setting, we pre-define a time series to graph mapping (e.g., Cosine similarity) and see if the model can predict the graph connectivity according to the historical time series. It makes sense because it has been proven that the vanilla ESN can predict the chaotic time series generated from nonlinear time-delay differential equations (e.g., Mackey-Glass).

In the experiment, we simulate a 5-node chaotic system. In total 10k chaotic time series data are simulated for each node as the "dynamics on graphs", with each time series having 500 timestamps.

Syn-Coupled

This synthetic dataset is for verifying if the proposed model can recover the graph structure from the nodes' time-series data under the assumption that the graph structure affects the interactions of time series. Random graphs are firstly generated with a pre-defined distribution. Each node of the graph generates base signals. The state of a node at a certain time point is determined by the node's base signal and the influence of its neighbors. When the graph is static, the signals on all the nodes will be synchronized. If we change the randomly generated graph structure every time l, the node signals will also be affected. Given the time series $S = S_1, S_2, ..., S_{n\times l}$ on all the nodes during $n \times l$, the experiment is to testify if the model can recover the graph structures $G = \{G_1, G_2, ..., G_n\}$

The dynamics of the weakly coupled time series which follow the governing equations:

$$\frac{d\theta_i}{dt} = \omega_i + \frac{K}{N} sin(\theta_j - \theta_i), \quad i = 1...N, \tag{B.2}$$

where θ_i is is the signal on node i. Runge-Kutta method is used for generating the time series according to (B.2)

We used the same external code[42] base used by NRI[39] for generating both Syn-Coupled-1 and Syn-Coupled-2. There are three initializing parameters for this implementation including the initial phase $Y0$, intrinsic freq W, coupling matrix K. When s is a random sample from a uniform distribution over $[0, 1)$, the parameter settings are described as following.

- $Y0$: All the nodes are set to $s*2*\pi$ in Syn-Coupled-1. The first $Y0$ is set to $[0, \pi, 0, 1, 5]$ for each nodes in Syn-Coupled-2, then $Y0$ is set to the last phase in the last time series segment for the following time series segments.
- W: W is set to $s * 9 + 1$ in Syn-Coupled-1. W is set to $[28, 19, 11, 9, 2]$ (the default values in the repository) in Syn-Coupled-2.
- K: The edges in both datasets follow the Bernoulli distribution $p = 0.5$.

In the experiment, we simulate a 5-node interacting system of coupled oscillators. In total 10k chaotic time series data are simulated for each node as the "dynamics on graphs", with each time series having 100 timestamps.

Brain

As an example application to biological data, we use the resting-state fMRI data from over 1000 subjects released by the Human Connectome Project [63]. The subjects included in the dataset underwent an in total one hours high-resolution T1-weighted anatomical MRI scan. The smallest units measured in the raw data were $2 \times 2 \times 2$ mm voxels. The temporal resolution was around 0.7s. The brain undergoes an anatomical parcellation, resulting in 68 gyral-based Regions of Interest (ROIs), evenly distributed with 34 cortical ROIs in each hemisphere. The brain activities were measured with Blood-oxygen-level-dependent imaging in fMRI for each voxel but are averaged to the 68 ROIs. The averaged BOLD signals represent the 1-dimensional time series on each ROI. Functional connectivity is defined as the correlation between the time series of BOLD data from various brain regions.

Forum

As an example application to social media data, we use the healthcare forum data used by Gao et al. [20]. This dataset was collected from the Breast Cancer Community and covers an eight-year period. The forum users' transition behavior patterns in different subforums are normally due to the changes in their health status and are reflected in their history activities. The users' transition behaviors can be modeled as sequences of transition graphs in which each node represents a subforum. In the experiment, we try to predict the users' accumulated transition graphs based on their history activity records.

Protein

As an example application to biology data, we use the protein folding data used in [2]. Protein folding is the natural process through which a protein chain attains its native 3-dimensional structure, typically a biologically functional conformation, in a rapid and consistent manner. During the protein folding process, the amino acids' position keeps changing. When the distance between two amino acid's distance is longer than a threshold, they can be viewed as unconnected. The connectivity of all the amino acids can form a dynamic graph. In the context of graph learning, this can be viewed as a graph comprising 8 nodes, where each node has attributes (x, y, z) representing the 3D coordinates of the atom in each amino acid. This results in 300 temporal graphs with a sequence length of 100.

B.2 B. Implementation Details and Experiment Settings

All experiments were conducted on a single 64-bit server with Nvidia TITAN V GPU.

B.2.1 Deep Echo-State Encoder

After the architecture of the ESN is optimized with NAS, the randomly generated ESN weights are normalized to meet a standard called Echo State Property (ESP) [35] as shown in Eq B.3. This is a common practice being used in almost all the ESNs.

$$\begin{aligned}
&\forall s = [u(1), u(2), ..., u(n)] \\
&\forall x, x' \in \mathbb{R}^{N_R} \text{ initial states} \\
&||\hat{\mathcal{R}}(s, x) - \hat{\mathcal{R}}(s, x')|| \to 0 \text{ as } n \to \infty
\end{aligned} \tag{B.3}$$

B.2.2 Dynamic Graph Topological Decoder

We tested message passing graph neural network (MPGNN) like in [39] as well as GAT as the dynamic graph topological decoder. In the experiment section, we only report the results with GAT but the results are very close with MPGNN.

The MPGNN module computes the graph adjacency/affinity matrix following the equation:

$$\begin{aligned}
&v \to e : h_{i,j}^{(1)} = f_e^{(1)}([h_i^{(1)}, h_j^{(1)}]) \\
&e \to v : h_j^{(2)} = f_v^{(2)}(\sum_{i \neq j} h_{i,j}^{(2)}) \\
&v \to e : h_{i,j}^{(2)} = f_e^{(2)}([h_i^{(2)}, h_j^{(2)}]) \\
&... \\
&output : A_{i,j} = f_{out}(e_{i,j}^{(n)})
\end{aligned} \tag{B.4}$$

here f are neural networks that map between the edge and node representations. We use fully-connected networks (MLPs) for the f functions.

Except for GAT and MPGNN, other options (e.g., MLP, RNN) can also be used for calculating pairwise node-to-node relationships in the last step and are compatible with our framework.

B.2.3 Deep-echo-state Architecture Optimization

As mentioned in Section 3.4, the search space is $A^{(R)} = [A^{(in)}, A]$, $A^{(R)} \in \mathbb{R}^{(d+R)\times R}$ where d is the input dimension and R is the size of the reservoir in ESN. In order to make the optimization possible with automatic differentiation, we relaxes the categorical edge choices to a continuous variable $A^{(R)}$ where $A^{(R)}_{i,j} := [0, 1]$. Similar approaches are used in NAS studies for image classification problems [30, 46]. The NAS stage, we randomly select a node and use only the first 10% of time series data for NAS. The bi-level optimization of our NAS formulation is to optimized for making auto-regression with ESN. The intuition is that the encoder works well and can be used for other downstream tasks if it can make accurate auto-regression. After the continuous $A^{(R)}$ is optimized, we sample the discrete according to the lowest threshold λ to meet the ESP as mentioned in Section B.2.1.

We tested different settings for the length of data used for NAS, ranging from 5% up to 100%. On most datasets, longer time series data don't help improving the encoder when the ratio reaches a threshold. We chose 10% as a fixed ratio for all the datasets as it's a sweet spot for all the datasets considering the performance and search time.