AUTOMATED ARCHITECTURE SYNTHESIS FOR ARBITRARILY STRUCTURED NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper introduces a fresh perspective on the architecture of Artificial Neural Networks (ANNs). Conventional ANNs generally employ predefined tree-like or Directed Acyclic Graph (DAG) structures for simplicity, yet such designs can constrain network collaboration and capacity due to limited horizontal and backward communication. By contrast, biological neural systems consist of billions of neurons interconnected in highly complex patterns, allowing each neuron to form context-dependent connections with others. Drawing inspiration from these biological systems, this work proposes a framework that learns to construct arbitrary graph structures during training. It also introduces "Neural Modules" as a way to group neural units, thereby facilitating communication among arbitrary nodes. Unlike traditional DAG-based ANNs, the proposed framework starts from complete graphs and permits unconstrained information exchange between neurons, more closely simulating the functioning of biological neural networks. Additionally, we develop a computational method for handling such arbitrary graph structures and a regularization strategy that organizes neurons into multiple independent and balanced Neural Modules. This organization helps mitigate overfitting and improves computational efficiency through parallelization. In summary, our approach enables ANNs to learn flexible, arbitrary structures that resemble those in biological systems. It demonstrates promising adaptability across different tasks and scenarios, and experimental results support its potential.

1 INTRODUCTION

This work presents a fresh perspective on the architecture of Artificial Neural Networks (ANNs). Conventional ANNs are typically organized in hierarchical, tree-like structures or Directed Acyclic Graphs (DAGs), either by manual design or through Neural Architecture Search (NAS) methods that operate within a DAG-based search space. However, such designs limit effective communication between nodes and introduce considerable structural bias.

From a theoretical standpoint, traditional DAG structures (including trees) can be viewed as imposing a topological ordering on nodes, where each node is only allowed to connect to its predecessors. In a DAG with p nodes, this results in at most

$$1 + 2 + ... + p = \frac{p(p+1)}{2} \tag{1}$$

possible edges. In practice, existing ANN architectures are optimized within this constrained space, which may restrict their expressive power and limit performance potential.

This study reconsiders traditional ANN design by noting that current connectivity schemes do not fully capture desirable properties of neural computation. Nodes in asynchronous tree-like layouts cannot form flexible connections, which impedes information flow and leads to inherent limitations. To address this, we propose a method for building synchronous graph structures using modular components called Neural Modules, which facilitate collaboration among neural units. In our framework, a graph with p nodes can have up to

$$p^2 \tag{2}$$

edges, thereby better leveraging the representational capacity of general graph topologies.

Such a structure may induce a collaborative effect among the neural units in the network. The graph framework can achieve superior performance both intuitively and theoretically, as elaborated earlier.

Prior work has recognized issues in contemporary ANN structures and attempted to introduce cyclic graph models. However, these studies did not thoroughly analyze the role of generalized cyclic structures in ANNs, nor did they provide a framework for automatically learning such architectures—akin to biological neural networks. These gaps form the core focus of our paper.

Our approach supports synchronous communication among all nodes in the structure and introduces a method for dynamically forming such structures during learning. These improvements enhance information transfer efficiency and increase the overall capacity of the neural architecture. By promoting node collaboration and supporting automatic structure learning, our method harnesses the collective potential of neural units in a manner closer to biological neural networks.

It should be noted that conventional tree-structured networks are a special case of our general graph formulation. We clarify the structural bias in existing ANN designs and provide a detailed analysis of integrating general graph structures into neural models in the appendix. Under our framework, multiple neural units cooperate to automatically implement specific functions during learning. Our goal is to help narrow the gap between current ANNs and more biologically-plausible generalized structures.

Designing such an architecture presents considerable challenges, including higher computational costs and increased risk of overfitting. To mitigate these issues, we introduce a novel regularization technique that organizes nodes into multiple independent Neural Modules. These modules can be processed in parallel on modern GPUs, support automatic node organization, improve learning efficiency, reduce overfitting, and lead to better overall performance.

Our learning framework shows strong adaptability across a variety of tasks. Experimental results on state-of-the-art networks indicate that our method achieves competitive performance in many real-world scenarios.

In summary, the main contributions of this work are:

1. An analysis of structural bias in existing tree-like neural networks, along with a detailed explanation of our proposed architectural enhancements.

2. A method that enables ANNs to automatically learn and construct arbitrary graph structures.

3. A novel regularization approach that organizes neural units into Neural Modules, improving structural efficiency via parallel computation and boosting performance by reducing overfitting.

2 Related Works

To advance the existing tree-like structure for NNs, Yuan (Yuan et al., 2020) recently provided a topological perspective, highlighting the benefits of dense connections enabled by shortcuts in optimization (Srivastava et al., 2015) (Sandler et al., 2018). Furthermore, sparsity constraints have been proven effective in optimizing learned structures across various applications (Srivastava et al., 2015) Chu et al. (2023) (Ahmed & Torresani, 2018) (He et al., 2016) (Huang et al., 2017). In these approaches, the structure of NNs is organized as a DAG, whereas our work organizes it into a more general graph structure.

Additionally, in recent years, the Cyclic Structure with the Forward-Forward Algorithm (Yang et al., 2024) has also attempted to design such a structure for NNs. The differences between our work and this study can be summarized as follows: First, the graph structure in (Yang et al., 2024) is predefined, whereas our framework automatically organizes the graph structure. Unlike predefined designs, our framework starts with a complete graph structure, where each neural unit has the potential to connect with any other neuron. Second, (Yang et al., 2024) achieves an equilibrium state through repetitive loops but does not explain the essence of the loop or the termination condition. In contrast, we conduct an in-depth analysis of the essence of the equilibrium state. Third, (Yang et al., 2024) does not analyze the size of cyclic graphs for the model or how to control them—factors that are crucial for the efficiency of the entire framework. For our framework, we propose NM regularization to control the structural complexity, thereby enhancing the model's performance in terms of both performance and efficiency. Detailed analysis can be found in the appendix.

The fixed point of the implicitly hidden layer can also serve as a solution (Bai et al., 2019) (Tsuchida & Ong, 2022) (Chu et al., 2023) Yang et al. (2022) Heaton et al. (2021) (Zucchet & Sacramento, 2022), as demonstrated in subsequent works (Bai et al., 2020) (Szekeres & Izsák, 2024) (Yang et al., 2023). Departing from the infinite structure of implicitly hidden layers (Chu et al., 2023), we organize the network into a general graph structure. Compared with implicitly hidden layers, our method improves efficiency through parallel computing and enhances performance by reducing overfitting.

Other structures, such as OptNet, integrate quadratic optimization problems for nodes within the same layer (Amos & Kolter, 2017) (Yan et al., 2021). However, this approach introduces additional bias.

Our work also involves NN compression. In recent years, various compression algorithms have been developed, including quantization (Han et al., 2020) (Shen et al., 2019) (He et al., 2018), low-rank approximation (Li & Shi, 2018) (Yang et al., 2019) (Yu et al., 2017), knowledge distillation (Kong et al., 2020) (You et al., 2018), and network pruning (Molchanov et al., 2019). In this paper, we seek to improve weight pruning for our framework using a method similar to (Lin et al., 2020), which evaluates the gradient of the pruned model and applies parameter updates to the dense model. In our framework, this process is coordinated with an elegant regularization method to automatically allocate Neural Modules. The detailed process is described in the algorithm provided in the appendix.

Graph Neural Networks (GNNs) are specifically designed to address the needs of geometric deep learning (Gori et al., 2005) (Fan et al., 2019) (Scarselli et al., 2008) (Abadal et al., 2021). GNNs adapt their structure to the input graph, capturing complex dependencies (Yong et al., 2007) (Abadal et al., 2021) (Fout et al., 2017) (Fan et al., 2019). Notably, GNNs primarily handle graphs as input data, which differs from our focus on constructing arbitrary graph structures for the network itself.

The flexibility of graph structures has also been explored in studies related to Reservoir Computing (Verstraeten et al., 2007) (Vargas & Zhang, 2023). These studies utilize a recurrent neural network framework where neuron connections are established randomly, and the weights remain static after initialization. In contrast, our Neural Module framework enables adaptive learning of both weights and network structure during processing.

Neural Architecture Search (NAS) has evolved from computationally expensive reinforcement learning and evolutionary methods to efficient gradient-based and weight-sharing approaches (Real et al., 2017) (Zoph & Le, 2017) (Pham et al., 2018) (Tan et al., 2019) (Zela et al., 2020) (Mellor et al., 2021). Modern methods like DARTS (Liu et al., 2019) use differentiable search. However, most NAS methods still optimize within a tree-like structure, limiting their structural flexibility.

3 METHODOLOGY

3.1 THE MATHEMATICAL FORMALIZATION OF THE MODEL

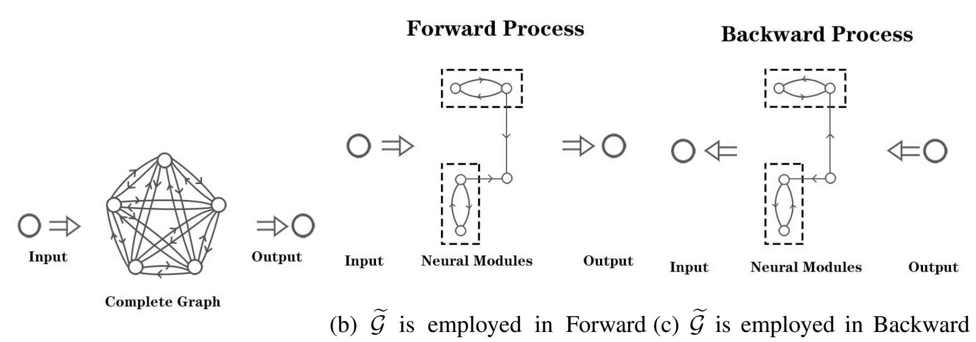
Let N^0 denote the input values fed into the input layer. Let N^m represent the nodes of the last layer, which feed into the output values. In our work, the intermediate structure is organized as a complete graph. The model is denoted by NMs, $NMs = \{N^0, E^1, \mathcal{G}, E^m, N^m\}$, where $\mathcal{G} = \{E, N\}$ and $n_i \in N$ is the ith node in $\mathcal{G}$, $e_{ij} \in E$ is the edge from n_i to n_j. Let the number of nodes in N be p, the number of nodes in N^0 be $|N^0|$, and the number of nodes in N^m be $|N^m|$.

3.2 MODEL STRUCTURE

In our framework, nodes are initially computed based on their input nodes, which solely distribute features. Additionally, each node is influenced by other nodes in the complete graph $\mathcal{G}$, resulting in mutual influence between nodes.

Our structure is constructed as follows: All intermediate nodes are organized into a general graph $\widetilde{\mathcal{G}}$ derived from a complete graph $\mathcal{G}$ where only edges with weights whose absolute values exceed a threshold γ are retained. In each iteration, the weights in $\mathcal{G}$ are updated, with corresponding adjustments made to $\widetilde{\mathcal{G}}$. Within this configuration, each node is influenced by all other nodes in the graph through the learning process. Nodes in the general graph $\widetilde{\mathcal{G}}$ are connected via directed edges with learnable weights. This mechanism allows each node to not only process its own input but also integrate information from other nodes.

In our framework, the architecture is generated by iteratively searching over a set of complete graphs and selecting crucial edges through pruning. The overall pipeline is depicted in Figure 1. In each iteration, the weights of the complete graph are updated as follows: following (Lin et al., 2020), we compute the gradients at the nodes of the current sparse graph $\widetilde{\mathcal{G}}$ from the preceding backward process and use them to update the weights of the dense underlying graph $\mathcal{G}$. Subsequently, $\mathcal{G}$ is pruned according to a parameter γ to obtain an updated sparse graph $\widetilde{\mathcal{G}}$. This updated $\widetilde{\mathcal{G}}$ is then employed in both the forward and backward processes of the subsequent iteration.



(a) Update $\mathcal{G}$ in each iteration. (b) $\widetilde{\mathcal{G}}$ is employed in Forward Process. (c) $\widetilde{\mathcal{G}}$ is employed in Backward Process.

Figure 1: An overview of the proposed framework's architecture.

In the following section, we elaborate on the process of calculating node values in the graph $\widetilde{\mathcal{G}}$.

3.3 Forward Process

In this paper, the value of each node is represented as x and the value of each edge is represented as w with corresponding node and edge indices. As introduced in the previous section, these values depend on both the nodes in N^0 and other nodes in $\widetilde{\mathcal{G}}$. Therefore, we need a synchronization method to handle this mutual dependence. We model this problem as a system of multivariate equations. For the values of the nodes in $\widetilde{\mathcal{G}}$, we have the following equations:

$$\begin{cases} w_{11} + \sum\limits_{j \neq 1} f(x_j) \cdot w_{j1} + \sum\limits_{j=1}^{|N^0|} x_j^0 \cdot w_{j1}^1 = x_1 \\ w_{22} + \sum\limits_{j \neq 2} f(x_j) \cdot w_{j2} + \sum\limits_{j=1}^{|N^0|} x_j^0 \cdot w_{j2}^1 = x_2 \\ \ldots \\ w_{pp} + \sum\limits_{j \neq p} f(x_j) \cdot w_{jp} + \sum\limits_{j=1}^{|N^0|} x_j^0 \cdot w_{jp}^1 = x_p \end{cases} \tag{3}$$

In the above equations, w_{11}, w_{22}, ..., w_{pp} are the weight of the self-loop edges in $\widetilde{\mathcal{G}}$ and represent the biases of the nodes. f is the activation function.

Let W^m be the weights of E^m and $X = \{x_1, x_2, ..., x_p\}$ be the values of the nodes in $\widetilde{\mathcal{G}}$. Then, the output values $\widetilde{Y} = X^m$ can be derived as $X^m = g(f(X) \cdot W^{mT})$, where g is the activation function for output.

Existing numerical methods, such as the Newton-Raphson method (Gawade, 2024), can effectively solve the above equations. In real-world applications, besides Newton's method, efficiency can be optimized using iterative methods, the dichotomy method, or the secant method—provided that the complexity of the coefficient matrix is well-controlled. Note that each variable is processed by the activation function f, making the transformation nonlinear. According to the Universal Approximation Theorem, there exists a graph that can effectively approximate any function. Further discussion of this point appears in the appendix.

3.4 Backward Process

To ensure correctness and consistency with the forward process, the gradient at each node during the backward process is also affected by the gradients of other nodes in $\widetilde{\mathcal{G}}$. Computing the gradient of the nodes in $\widetilde{\mathcal{G}}$, we consider the gradient of the output layer as $\nabla X^m = \nabla Y$. Similar to the forward process, the gradients of the nodes also interact with each other. Note that each node has been processed by the activation function. Thus, we model the gradients of the nodes in the graph as variables in the following system of equations:

$$\begin{cases} \sum_{j \neq 1} \nabla x_j \cdot f'(x_j) \cdot w_{1j} + \sum_{j=1}^{|N^m|} \nabla x_j^m \cdot g'(x_j^m) \cdot w_{1j}^m = \nabla x_1 \\ \sum_{j \neq 2} \nabla x_j \cdot f'(x_j) \cdot w_{2j} + \sum_{j=1}^{|N^m|} \nabla x_j^m \cdot g'(x_j^m) \cdot w_{2j}^m = \nabla x_2 \\ \ldots \\ \sum_{j \neq p} \nabla x_j \cdot f'(x_j) \cdot w_{pj} + \sum_{j=1}^{|N^m|} \nabla x_j^m \cdot g'(x_j^m) \cdot w_{pj}^m = \nabla x_p \end{cases} \tag{4}$$

Finally, we can compute the gradients for the edges in the complete graph $\mathcal{G}$ according to the gradient of the nodes in $\widetilde{\mathcal{G}}$ as (Lin et al., 2020).

Based on the system of equations, we first consider the gradient at each node. For the jth node in the graph, the weights of its incoming edges correspond to the jth($1 \leq j \leq p$) column of the weight matrix in $\mathcal{G}$. Since graph data does not have a layered structure, we introduce the following operator for each node to represent its neighboring nodes:

$$\mathcal{H}_j = [f(x_1), ..., f(x_{j-1}), 1, f(x_{j+1}), ..., f(x_p)] \ , \tag{5}$$

which is derived from the system of equations in the forward pass. Then, using the gradient at the jth node, its corresponding gradient for $W_{:j}^T, 1 \leq j \leq p$ in $\mathcal{G}$ can be formulated as follows:

$$\nabla W_{:j}^T = \nabla x_j \circ f'(x_j) \cdot \mathcal{H}_j \ . \tag{6}$$

Second, for the gradient for the edges in E^m, according to the backward process,

$$\nabla W^m = \nabla X^{mT} \circ g'(X^{mT}) \cdot f(X) \ . \tag{7}$$

Third, for the gradient for the edges in E^1, according to the backward process,

$$\nabla W^1 = \nabla X^T \circ f'(X^T) \cdot X^0 \ . \tag{8}$$

At last, the standard update process for gradient-based algorithms is executed.

3.5 Neural Module Optimization

In this section, we formalize the concept of Neural Modules (NM): **A Neural Module is defined as a Strongly Connected Component (SCC) of $\widetilde{\mathcal{G}}$.**

Directly solving Equations (3) and (4) for large networks is challenging. Note that they constructed these equations on a general graph $\widetilde{\mathcal{G}}$. We then compute the solution asynchronously over $\widetilde{\mathcal{G}}$ using a generalized topological sort. However, when encountering synchronously structured Neural Modules, we solve the corresponding equation system directly as the dashed box in Figure 1, as its scale remains manageable due to the modules' limited size.

First, we introduce the previously mentioned NM regularization. For a general graph $\widetilde{\mathcal{G}}$, we first normalize the absolute value of its adjacency matrix to obtain $\widetilde{W}$. We then define its distance matrix D, where each element $d_{ij} \in D$ is defined as :

$$d_{ij} = e^{-\widetilde{w}_{ij}}. \tag{9}$$

Second, NM regularization takes into account the number of nodes in each Neural Module. We introduce the operator $\mathcal{Z} = [z_1, z_2, ..., z_p]$, where each element $z_i, 1 \leq i \leq p$ in $\mathcal{Z}$, z_i represents the number of nodes in the Neural Module corresponding to node n_i.

Based on the inverse proportionality law in two-dimensional graph space, we introduce the repulsion matrix R for $\widetilde{\mathcal{G}}$. Each element $r_{ij} \in R$ is defined as

$$r_{ij} = \frac{z_i * z_j}{d_{ij}} \tag{10}$$

Let α be the regularization parameter. Through NM regularization, the repulsion matrix R adaptively adjusts α in each iteration. This process facilitates the automatic organization of the graph into balanced, appropriately sized subgraphs, forming rational Neural Modules that effectively utilize neural units. For the ith node in $\widetilde{\mathcal{G}}$, $1 \leq i \leq p$, our NM regularization is formulated as:

$$J_{NM}(x_{:i}) = J(x_{:i}) + \alpha \sum_{j=1}^{p} r_{ji} \cdot w_{ji}^2, \tag{11}$$

where J denotes the objective function.

During the backpropagation process, the weight of each edge is updated as follows:

Theorem 3.1. *For NM regularization, in each iteration with learning rate η, parameter w_{ij} upgrades as follow:*

$$w_{ij} \leftarrow w_{ij}(1 - \eta\alpha r_{ij}) - \eta \frac{\partial J}{\partial w_{ij}} \tag{12}$$

Thus, when r_{ij} takes a higher value, w_{ij} is more likely to approach zero.

To simplify the analysis and enhance understanding of NM regularization, we propose a theorem by omitting the activation function and analyzing its effect under linear regression:

Theorem 3.2. *Under linear regression with $y = Xw_{:i} + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \delta^2 I)$ and $w_{:i} \sim \mathcal{N}(0, \tau^2 diag(r_{:i})^{-1})$, for NM regularization, the weight distribution is given by:*

$$w_{:i} \sim \mathcal{N}((X^T X + \lambda diag(r_{:i}))^{-1} X^T y, \delta^2 (X^T X + \lambda diag(r_{:i}))^{-1}), \tag{13}$$

where $\lambda = \frac{\delta^2}{\tau^2}$.

From the above theorem, a larger repulsion term in R brings the expectation of the weight closer to zero and reduces its variance.

In NM regularization, for each element r_{ij} in the repulsion matrix R, a larger z_i or z_j increases its value, resulting in a stronger repulsive force. Similarly, smaller elements in the distance matrix D have the same effect. These effects can be summarized in two aspects:

Between Neural Modules: NM regularization prevents the formation of excessively large Neural Modules, as analyzed earlier.

Within Neural Modules: NM regularization helps avoid overly complex adjacency matrices. This is critical because overly complex adjacency matrices can lead to solution instability within Neural Modules—often caused by coefficient matrices with high condition numbers.

At last, we talk about the convergence of NM regularization. Let w^* be the parameters of the global optimal model, w_0 be the parameters of the original model, and $\tilde{w_0}$ be the parameters of the optimized model in the first iteration. To provide convergence guarantees for NM regularization, we establish theoretical bounds based on the following assumptions: The training objective is smooth, satisfying $\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|, \forall w, v \in \mathbb{R}^p$, for some constant $L > 0$. The stochastic gradients are bounded, with $\mathbb{E}\|\nabla w\|^2 \leq G^2$. Under these assumptions, the convergence of NM regularization is presented in the following theorem:

Theorem 3.3. *Let the learning rate be $\zeta = \frac{c}{\sqrt{T}}$, where $c = \sqrt{\frac{f(w_0) - f(w^*)}{LG^2}}$ and T is the number of iterations. For a pruned model selected according to the definition of Neural Modules, the following inequality holds in expectation over the selected edges u as $\widetilde{\mathcal{G}}$:*

$$\mathbb{E}\|\nabla u\|^2 = \mathcal{O}(\sqrt{\frac{L(f(w_0) - f(w^*))}{T}} G + L^2 \|w_0 - \tilde{w_0}\|^2). \tag{14}$$

3.6 Algorithm

As introduced before, directly solving Equations 3 and 4 for $\widetilde{\mathcal{G}}$ is computationally infeasible. Leveraging NM regularization, our algorithm instead solves these two equations for each Neural Module individually.

In this section, we initialize the adjacency matrix of the complete graph $\mathcal{G}$ as $\mathcal{K}$. Then, approximate $\mathcal{K}$ to the coefficient matrix $\mathcal{C}$ from $\widetilde{\mathcal{G}}$, eliminating all edges whose weights have absolute values lower than γ.

The computation of the graph is based on a method similar to topological sorting. For each Neural Module NM_i, $InArc(NM_i)$ denotes the number of edges e_{js} in $\mathcal{C}$ that connect from node $n_j \notin NM_i$ to node $n_s \in NM_i$. Conversely, $OutArc(NM_i)$ represents the number of edges e_{js} in $\mathcal{C}$ that connect from node $n_j \in NM_i$ to node $n_s \notin NM_i$. Here, $|NM|$ denotes the number of nodes within the Neural Module. The complete algorithm for this process is outlined in the Appendix.

In our algorithm, we employ Tarjan's algorithm (Tarjan, 1972) to identify strongly connected components, which form the basis for constructing Neural Modules. Tarjan's algorithm is the classic method for detecting strongly connected components. While other candidate algorithms exist for this problem, none achieve a lower time complexity. Furthermore, for very large graphs, we can adopt the Pregel model (Malewicz et al., 2010) to handle the task in a distributed manner.

The algorithm is based on a generalized form of topological sorting. Each iteration consists of two main processes: The first process involves propagation according to the traditional NN forward process. The second process aims to handle Neural Modules as a system of equations in parallel. During each iteration, the algorithm updates by removing irrelevant edges. In summary, our framework is detailed in Algorithm 1, Algorithm 2, and Algorithm 3, which are provided in the appendix.

For Algorithm 1, the complexity analysis of the main part is as follows. The complexity of Tarjan's algorithm is $\mathcal{O}(|N| + |E|)$, and the complexity of checking InArc or OutArc is $\mathcal{O}(|E|)$. The complexity of solving the system of equations can be optimized to $\mathcal{O}(|NM|^2)$, where the complexity for each Neural Module (NM) has been reduced through NM regularization as introduced in **Between Neural Modules** and **Within Neural Modules**. Therefore, in the worst-case scenario—where the structure is organized as a linear chain that prevents parallel computing—the overall complexity of our propagation algorithm is $\mathcal{O}(|N| + |E| + s * |NM|^2)$ (where s is the number of Neural Modules). If the topology supports parallel computing, the overall complexity can be optimized to $\mathcal{O}(|N| + |E| + max(|NM|^2))$.

4 Experiments

4.1 Performance Evaluation of Neural Modules Beyond Traditional Tree-like Structures

In this section, we present experiments conducted using our Neural Modules, comparing their performance with traditional NN methods and several state-of-the-art models that go beyond traditional tree-like structures. These baselines include implicit hidden layers (DEQ), a topological approach that models NNs as DAGs, and the recently introduced OPTNET, as introduced in the Related Work section. The results are presented in table 1 for four real-world datasets.

The four datasets used in our experiments correspond to different real-life scenarios. Codon Dataset: Comprises codon usage frequencies in genomic coding DNA from a diverse sample of organisms across different taxa, obtained from the CUTG database. Facebook Large Page-Page Network Dataset: Contains a webgraph of verified Facebook page-page connections. Daily and Sports Activities Dataset: Includes motion sensor data of 19 daily and sports activities, each performed by 8 subjects in their own style for 5 minutes. Gas Sensor Dataset: Consists of measurements from 16 chemical sensors exposed to six different gases at various concentration levels. All these tasks are classification problems, and we evaluate performance based on the **error rate** of each algorithm.

We compared the performance of our framework with baselines that go beyond traditional tree-like structures (e.g., DEQ and OPTNET as introduced in the related work section) for neural networks of varying sizes. Through analysis of the experimental results, we can validate our claim that our framework exhibits significant advantages at this scale.

We assessed the effectiveness of our NMs and other methods across different node complexities to understand how NMs perform under varying levels of complexity. The nodes were initially organized using NN, DEQ, DAG, and OPTNET, and their error rates were recorded. Our experiments also compared the performance of NMs with two regularization strategies: the commonly used L2 regularization and the proposed NM regularization. The results, presented in Table 1, show that our novel structure consistently achieves superior performance in most cases. Additionally, NM regularization outperforms the baseline L2 regularization in most scenarios. With an optimal number of nodes, our NM regularization achieves the best performance across all datasets.

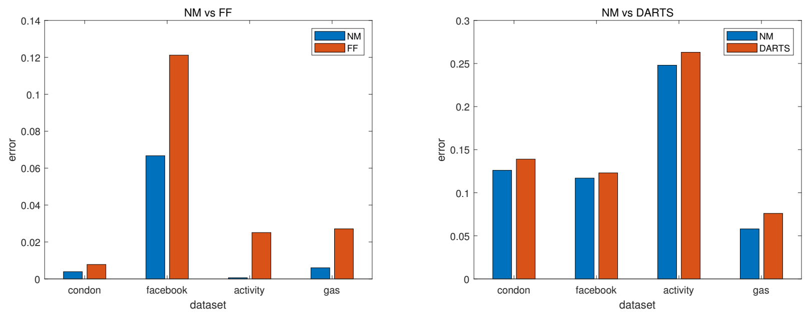
From Table 1, we can conclude that NNs consistently exhibit improved performance when nodes are organized into Neural Modules. The performance of our NMs can be further enhanced through regularization, as explained earlier. Note that methods such as DEQ, DAG, and OPTNET may face scalability issues or high computational overhead on large graphs, primarily because they were originally designed for problems at the layer scale and have not been extensively optimized for graph-level tasks.

Table 1: The performance of algorithms

				Condon			
Methods	80	100	200	300	500	1000	3000
NN	0.151 ± 0.004	0.152 ± 0.005	0.133 ± 0.008	0.130 ± 0.006	0.180 ± 0.004	0.159 ± 0.008	0.180 ± 0.006
DEQ	0.147 ± 0.002	0.139 ± 0.004	0.128 ± 0.005	0.221 ± 0.003	0.275 ± 0.004	over time	over time
DAG	0.186 ± 0.003	0.167 ± 0.003	0.150 ± 0.006	0.172 ± 0.005	0.213 ± 0.006	over time	over time
OPTNET	$\mathbf{0.140 \pm 0.003}$	0.150 ± 0.004	0.165 ± 0.004	0.143 ± 0.006	0.166 ± 0.003	over time	over time
NMs	0.157 ± 0.006	0.142 ± 0.004	0.127 ± 0.004	0.129 ± 0.003	0.155 ± 0.002	0.160 ± 0.005	$0.0.179 \pm 0.005$
NMsL2	0.158 ± 0.004	0.136 ± 0.006	0.128 ± 0.008	0.128 ± 0.003	$\mathbf{0.150 \pm 0.002}$	0.159 ± 0.004	0.178 ± 0.003
NMsNM	0.155 ± 0.002	$\mathbf{0.135 \pm 0.003}$	$\mathbf{0.126 \pm 0.004}$	$\mathbf{0.127 \pm 0.005}$	0.151 ± 0.004	$\mathbf{0.158 \pm 0.006}$	$\mathbf{0.178 \pm 0.003}$
				Activity			
Methods	80	100	200	300	500	1000	3000
NN	0.379 ± 0.006	0.354 ± 0.008	0.316 ± 0.007	0.304 ± 0.006	0.295 ± 0.005	0.309 ± 0.004	0.290 ± 0.004
DEQ	0.382 ± 0.004	0.364 ± 0.003	0.332 ± 0.008	0.321 ± 0.004	0.275 ± 0.003	over time	over time
DAG	0.639 ± 0.005	0.624 ± 0.005	0.538 ± 0.005	0.538 ± 0.003	0.513 ± 0.003	over time	over time
OPTNET	0.382 ± 0.006	0.364 ± 0.005	0.342 ± 0.006	0.363 ± 0.003	0.407 ± 0.005	over time	over time
NMs	0.350 ± 0.004	0.332 ± 0.007	0.308 ± 0.006	0.264 ± 0.002	0.263 ± 0.007	0.306 ± 0.006	0.300 ± 0.008
NMs&L2	$\mathbf{0.348 \pm 0.003}$	0.320 ± 0.003	0.288 ± 0.003	0.257 ± 0.002	0.260 ± 0.006	$\mathbf{0.295 \pm 0.005}$	0.293 ± 0.004
NMs&NM	0.364 ± 0.003	$\mathbf{0.314 \pm 0.003}$	$\mathbf{0.284 \pm 0.003}$	$\mathbf{0.250 \pm 0.003}$	$\mathbf{0.248 \pm 0.004}$	0.298 ± 0.004	$\mathbf{0.287 \pm 0.003}$
				Facebook			
Methods	80	100	200	300	500	1000	3000
NN	0.140 ± 0.006	0.130 ± 0.008	0.131 ± 0.006	0.160 ± 0.006	0.208 ± 0.006	0.167 ± 0.008	0.166 ± 0.005
DEQ	0.147 ± 0.006	0.121 ± 0.006	0.126 ± 0.005	0.169 ± 0.005	0.447 ± 0.006	over time	over time
DAG	0.168 ± 0.007	0.139 ± 0.007	0.148 ± 0.004	0.159 ± 0.005	0.647 ± 0.007	over time	over time
OPTNET	0.173 ± 0.004	0.144 ± 0.005	0.158 ± 0.003	0.225 ± 0.006	0.193 ± 0.005	over time	over time
NMs	0.135 ± 0.003	0.120 ± 0.004	$\mathbf{0.126 \pm 0.003}$	0.149 ± 0.004	0.169 ± 0.005	0.170 ± 0.003	0.168 ± 0.004
NMs&L2	0.135 ± 0.002	0.120 ± 0.003	0.137 ± 0.003	0.150 ± 0.003	0.165 ± 0.006	0.167 ± 0.004	0.163 ± 0.0048
NMs&NM	$\mathbf{0.134 \pm 0.00}4$	$\mathbf{0.117 \pm 0.003}$	0.129 ± 0.004	$\mathbf{0.145 \pm 0.004}$	$\mathbf{0.164 \pm 0.006}$	$\mathbf{0.166 \pm 0.005}$	$\mathbf{0.162 \pm 0.006}$
				Gas			
Methods	80	100	200	300	500	1000	3000
NN	0.073 ± 0.006	0.087 ± 0.005	$\mathbf{0.090 \pm 0.003}$	0.207 ± 0.006	0.089 ± 0.006	0.138 ± 0.007	0.112 ± 0.004
DEQ	0.102 ± 0.008	0.102 ± 0.007	0.138 ± 0.003	0.169 ± 0.007	0.447 ± 0.008	over time	over time
DAG	0.118 ± 0.004	0.084 ± 0.005	0.321 ± 0.004	0.239 ± 0.008	0.160 ± 0.004	over time	over time
OPTNET	0.063 ± 0.002	0.105 ± 0.002	0.105 ± 0.003	$\mathbf{0.103 \pm 0.003}$	0.160 ± 0.006	over time	over time
NMs	0.064 ± 0.003	0.082 ± 0.004	0.157 ± 0.002	0.126 ± 0.005	$\mathbf{0.087 \pm 0.003}$	0.140 ± 0.004	0.116 ± 0.008
NMs&L2	0.060 ± 0.006	0.081 ± 0.004	0.165 ± 0.004	0.120 ± 0.004	0.089 ± 0.003	0.136 ± 0.003	0.110 ± 0.007
NMs&NM	$\mathbf{0.058 \pm 0.006}$	$\mathbf{0.075 \pm 0.003}$	0.154 ± 0.006	0.120 ± 0.003	0.088 ± 0.003	$\mathbf{0.132 \pm 0.003}$	$\mathbf{0.108 \pm 0.005}$

4.2 Performance of Neural Modules with FF and DARTS

To validate our approach, we conducted comparative experiments using the Cyclic Forward-Forward algorithm (Yang et al., 2024),by formulating the multi-class classification task as a binary classification task on the four datasets. We also compared our method with the well-known NAS method DARTS (Liu et al., 2019). As shown in Figure 2, our Neural Module (NM) framework achieves higher accuracy than both the Cyclic Architecture with the Forward-Forward Algorithm and Differential Architecture Search (DARTS).



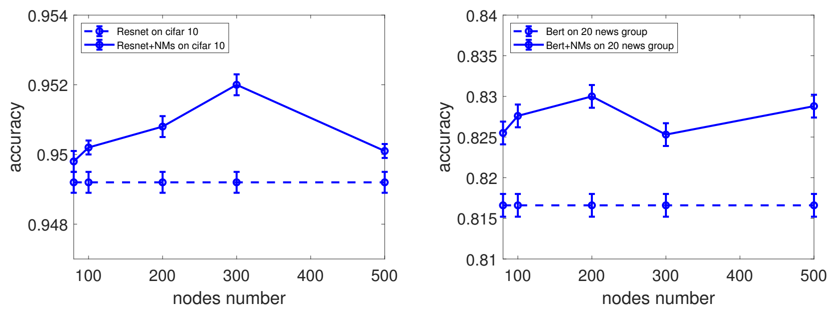
(a) Neural Modules vs FF (b) Neural Modules vs DARTS

Figure 2: It can be observed that our model achieves a lower error rate than both the Cyclic Forward-Forward algorithm and DARTS.

4.3 Performance of Neural Modules with Modern Structures

Furthermore, to demonstrate that our Neural Modules (NMs) can be integrated into modern architectures and scale to larger problems, we evaluated their performance on CIFAR-10 and the 20 Newsgroups dataset. For CIFAR-10, we embedded our NMs within a standard ResNet by replacing its fully connected layers with our graph-based structure. As illustrated in Figure 3a, this integration led to a measurable improvement in accuracy. Similarly, for the 20 Newsgroups dataset, we incorporated the NMs into a Transformer architecture, again substituting the standard fully connected components with our design. Figure 3b shows that this adaptation also yielded a substantial gain in performance. Together, these results confirm that our method can be effectively combined with contemporary architectures and enhances performance across different domains.

It is important to note that the purpose of these experiments was not to achieve state-of-the-art **accuracy** on these benchmarks, nor were the models heavily fine-tuned. Instead, our goal was to obtain a reasonable baseline that demonstrates the capability of our framework to integrate seamlessly into modern architectures and to handle diverse, realistic problems. The consistent improvements observed validate this objective.



(a) Performance on Cifar 10 (b) Performance on 20 News Group

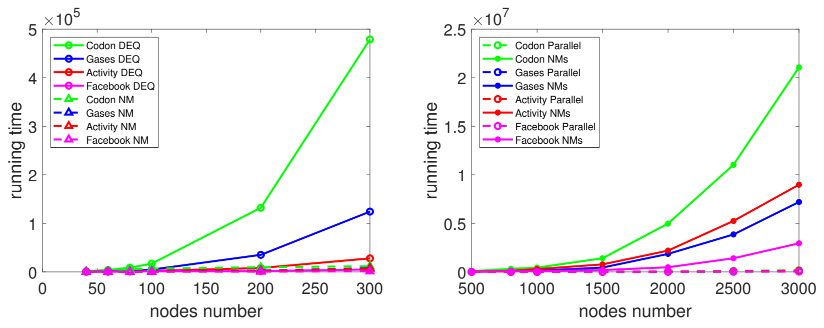
Figure 3: Collectively, these results validate that our method is not only compatible with modern architectures but also consistently enhances performance across diverse tasks.

4.4 Efficiency of Neural Modules

In this section, we focus on optimizing the efficiency of Neural Modules through the application of NM regularization. As mentioned earlier, all considered structures are subgraphs of a fully connected graph. Our NM regularization serves as a powerful mechanism for structural optimization, enhancing the effectiveness and balance of Neural Modules.

To evaluate the efficiency of our NM regularization, we compared the model's running time across different complexities (represented by node counts below 300). Figure 4a shows that the efficiency of NM regularization significantly outperforms DEQ, especially when the number of nodes is larger. This superiority is attributed to NM regularization's ability to create multiple independent Neural Modules, which effectively reduce computational complexity.

For networks with a larger number of nodes, we can leverage the parallel processing capabilities of Neural Modules to further enhance the efficiency of our framework, as discussed earlier. NM regularization facilitates the creation of multiple independent and well-balanced Neural Modules, which are inherently suitable for parallel computing—particularly when using GPU acceleration. In this extension, we increased the node count from 300 to 3000 and incorporated GPU hardware acceleration to compute the algorithms more efficiently.



(a) Nodes' number up to 300. Neural modules vs.DEQ.

(b) Nodes' number up to 3000. NM parallel vs. Neural modules.

Figure 4: The efficiency of NM regularization. Measured by seconds.

In Figure 4b, we compared the running time of Neural Modules operating in parallel with that of Neural Modules without parallelization for large-scale models. For these experiments, we used 12 threads. Note that Neural Modules without parallel computing would exceed the time limit for larger node counts, so we approximated the running time using partial data. Our results indicate that the parallel implementation of NM regularization achieves a computational speedup of approximately 10 times. This demonstrates the substantial efficiency gains achievable through parallel processing in the context of NM regularization.

Note that in this experiment, we use a lower parameter γ to form more complex neural modules for evaluating the progress of efficiency. However, to verify its performance, we adopt much simpler neural modules to avoid overfitting.

From these experiments, it is evident that our Neural Module framework can significantly enhance the performance of Neural Networks. By introducing additional parameters (as analyzed in previous sections), our framework enables the Neural Network structure to explore a nearly complete search space, effectively reducing the bias associated with current tree-like structures and achieving much better performance. Moreover, the NM regularization and parallel computing techniques we introduced further empower our method, allowing it to be effectively applied to larger networks.

5 Conclusion

This study introduces a novel general graph structure for NNs, aiming to improve performance by enabling efficient information transfer. We analyze the structural bias of current tree-like structures and propose a synchronization method for the simultaneous calculation of node values, thereby fostering collaboration within Neural Modules. Additionally, we propose a novel NM regularization method that encourages the learned structure to prioritize critical connections and automatically form multiple independent, balanced neural structures—facilitating more efficient computation through parallel processing. This approach not only reduces the computational load associated with managing a large number of nodes but also improves performance by mitigating overfitting. Quantitative experimental results confirm that our proposed method outperforms traditional NN structures.

REFERENCES

Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)*, 54(9):1–38, 2021.

Karim Ahmed and Lorenzo Torresani. Maskconnect: Connectivity learning by gradient descent. In *European Conference on Computer Vision (ECCV)*, 2018.

Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning (ICML)*, pp. 136–145, 2017.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:5238–5250, 2020.

Haoyu Chu, Shikui Wei, and Ting Liu. Learning robust deep equilibrium models. *arXiv preprint arXiv:2304.12707*, 2023.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference (WWW)*, pp. 417–426, 2019.

Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Shashank Gawade. The newton-raphson method: A detailed analysis. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 12(11):729–734, 2024.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, volume 2, pp. 729–734, 2005.

Kai Han, Yunhe Wang, Yixing Xu, Chunjing Xu, Enhua Wu, and Chang Xu. Training binary neural networks through learning with noisy supervision. In *International Conference on Machine Learning (ICML)*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Howard Heaton, Samy Wu Fung, Aviv Gibali, and Wotao Yin. Feasibility-based fixed point networks. *Fixed Point Theory and Algorithms for Sciences and Engineering*, 2021(1):1–19, 2021.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Shumin Kong, Tianyu Guo, Shan You, and Chang Xu. Learning student networks with few data. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Chong Li and C. J. Richard Shi. Constrained optimization based low-rank approximation of deep neural networks. In *European Conference on Computer Vision (ECCV)*, 2018.

Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations (ICLR)*, 2020.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *ACM SIGMOD International Conference on Management of Data*, pp. 135–146, 2010.

Joe Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training. In *International Conference on Machine Learning (ICML)*, 2021.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Hieu Pham, Melody Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, 2018.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning (ICML)*, 2017.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Mingzhu Shen, Kai Han, Chunjing Xu, and Yunhe Wang. Searching for accurate binary neural architectures. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 0–0, 2019.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Béla J. Szekeres and Ferenc Izsák. On the computation of the gradient in implicit neural networks. *The Journal of Supercomputing*, 2024.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1 (2):146–160, 1972.

Russell Tsuchida and Cheng Soon Ong. When are equilibrium networks scoring algorithms? In *NeurIPS 2022 Workshop on Score-Based Methods*, 2022.

Danilo Vasconcellos Vargas and Heng Zhang. A survey on reservoir computing and its interdisciplinary applications beyond traditional machine learning. *arXiv preprint arXiv:2307.15092*, 2023.

David Verstraeten, Benjamin Schrauwen, and Jan Van Campenhout. An overview of reservoir computing: Theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN)*, pp. 471–482, 2007.

Kai Yan, Jie Yan, Chuan Luo, Liting Chen, Qingwei Lin, and Dongmei Zhang. A surrogate objective framework for prediction+optimization with soft constraints. In *NeurIPS*, 2021.

Liang Wei Yang, Hengrui Zhang, Weizhi Zhang, Zihe Song, Jing Ma, Jiawei Zhang, and Philip S. Yu. Beyond directed acyclic computation graph with cyclic neural network. *arXiv preprint arXiv:2402.13041*, 2024.

Zhaohui Yang, Yunhe Wang, Chuanjian Liu, Hanting Chen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Legonet: Efficient convolutional neural networks with lego filters. In *International Conference on Machine Learning (ICML)*, 2019.

Zonghan Yang, Tianyu Pang, and Yang Liu. A closer look at the adversarial robustness of deep equilibrium models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:10448–10461, 2022.

Zonghan Yang, Peng Li, Tianyu Pang, and Yang Liu. Improving adversarial robustness of deep equilibrium models with explicit regulations along the neural dynamics. In *International Conference on Machine Learning (ICML)*, 2023.

Sweah Liang Yong, Markus Hagenbuchner, Ah Chung Tsoi, Franco Scarselli, and Marco Gori. Document mining using graph neural network. In *International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, pp. 458–472, 2007.

Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning with single-teacher multi-student. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Kun Yuan, Quanquan Li, Jing Shao, and Junjie Yan. Learning connectivity of neural networks from a topological perspective. In *European Conference on Computer Vision (ECCV)*, 2020.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Nicolas Zucchet and João Sacramento. Beyond backpropagation: Bilevel optimization through implicit differentiation and equilibrium propagation. *Neural Computation*, 34(12):2309–2346, 2022.

A The Bias of the Traditional Tree-like Structure

In this section, we demonstrate that the existing tree-like neural network (NN) structure is essentially a special case of the framework we propose for solving systems of equations. For a tree-like structure with m levels, let X^i denote the intermediate values at the ith level and X^0 represent the input values. For brevity, biases are omitted here. The asynchronous computation process of the current tree-like structure can be formalized within our framework as the following system of equations:

$$\begin{cases} X^0 \cdot W^{1T} = X^1 \\ X^1 \cdot W^{2T} = X^2 \\ ... \\ X^{m-1} \cdot W^{mT} = X^m. \end{cases} \tag{15}$$

By formatting the inputs and the values of all nodes in the neural network into a variable vector $\mathcal{X} = (X^0, X^1, X^2, ..., X^m)$, the above system can be simplified to $\mathcal{X} \cdot \mathcal{C}^T = 0$.

Furthermore, the coefficient matrix $\mathcal{C}$ for the tree-like structure is structured as Figure 5.

$$\begin{pmatrix} W^1 & -E^1 & 0 & 0 & ... & 0 & 0 \\ 0 & W^2 & -E^2 & 0 & ... & 0 & 0 \\ 0 & 0 & W^3 & -E^3 & ... & 0 & 0 \\ ...... \\ 0 & 0 & 0 & 0 & ... & W^m & E^m \end{pmatrix}$$

Figure 5: The Coefficient $\mathcal{C}$ for Tree-like Structure

Here E^i denotes the identity matrix corresponding to the ith level

From this system of equations, it is evident that the traditional NN structure constitutes a special case of our equation system, which can be solved asynchronously. In the tree-like structure, the coefficient matrix is composed of the parameter matrices W^i for each level, with these parameters positioned near the diagonal of the matrix.

Notably, the tree-like structure imposes stricter constraints on each node within the network. Specifically, if nodes in a tree structure depend exclusively on neurons from the immediately preceding layer, their capacity to approximate complex functions is severely limited.

To address this limitation, some existing works have generalized the tree-like structure to Directed Acyclic Graphs (DAGs)—as exemplified by ResNet. In such architectures, additional weights (denoted as V^k for the kth level) are introduced to the lower triangular region of the coefficient matrix $\mathcal{C}$. The updated matrix $\mathcal{C}$ is structured as Figure 6.

$$\begin{pmatrix} W^1 & -E^1 & 0 & 0 & ... & 0 & 0 \\ V^1 & W^2 & -E^2 & 0 & ... & 0 & 0 \\ V^1 & V^2 & W^3 & -E^3 & ... & 0 & 0 \\ ...... \\ V^1 & V^2 & V^3 & V^4 & ... & W^m & E^m \end{pmatrix}$$

Figure 6: The Coefficient Matrix $\mathcal{C}$ for DAG(e.g. Resnet)

This modification enables more flexible and expressive models, which can better approximate complex functions and handle larger datasets.

While lower triangular coefficient matrices typically represent the constraint of strictly causal (asynchronous) structures, the upper triangular region remains largely untapped. In this work, we extend such structures to synchronous ones by relaxing the constraint of strict lower triangularity, generalizing the coefficient matrix $\mathcal{C}$ to a full adjacency matrix that can represent arbitrary directed graphs. This allows the construction of more densely interconnected networks, offering potential for greater flexibility. Consequently, our framework moves neural architectures from strictly tree-like, causal dependencies toward more general graph-based structures.

B THE RATIONALE OF INTRODUCING NEURAL MODULES

For traditional tree-like structures, the asynchronous forward and backward propagation processes can also be interpreted as solving a system of equations. In our framework, by contrast, the NN structure is modeled as a general graph with p nodes.

A key design choice in our model is that the diagonal elements of the network's adjacency matrix correspond to the bias of each node. During formulation, input- and bias-related terms are assigned to the right-hand side of the equation system, while terms associated with node values are placed on the left-hand side. After each neural unit is processed using the input X^0, the coefficient matrix $\mathcal{C}$ takes the form shown as Figure 7.

$$\begin{pmatrix} -1 & w_{2,1} & w_{3,1} & w_{4,1} & ... & w_{p-1,1} & w_{p,1} \\ w_{1,2} & -1 & w_{3,2} & w_{4,2} & ... & w_{p-1,2} & w_{p,2} \\ w_{1,3} & w_{2,3} & -1 & w_{4,3} & ... & w_{p-1,3} & w_{p,3} \\ w_{1,4} & w_{2,4} & w_{3,4} & -1 & ... & w_{p-1,4} & w_{p,4} \\ ...... \\ w_{1,p} & w_{2,p} & w_{3,p} & w_{4,p} & ... & w_{p-1,p} & -1 \end{pmatrix}$$

Figure 7: The Coefficient Matrix $\mathcal{C}$ for Our General Graph Structure

A detailed breakdown of the coefficient matrix's role in computation is provided in the ”Forward Process” subsection (Section 3.3).

By design, our framework enhances the representational capacity of individual neurons, unlocking the full potential of neural networks. Concurrently, it eliminates the structural bias inherent in predefined architectures—such as traditional tree-like structures or DAGs. This innovation makes our framework more adaptable and less constrained by fixed architectural biases, resulting in more flexible and effective NN designs.

Solving systems of equations with large coefficient matrices $\mathcal{C}$ can be computationally challenging. To address this, our framework introduces Neural Module (NM) Regularization: an approximation method for $\mathcal{C}$ that integrates parallel computation to improve efficiency.

C CONNECTION WITH DEEP EQUILIBRIUM MODELS (DEQ)

Prior research has identified the existence of equilibrium states in infinitely deep weight-tied neural networks. Deep Equilibrium Models (DEQ) formalize this concept by modeling the forward pass as finding a fixed point of an implicit layer, effectively solving for the root of an implicit equation. In this work, we show that the dynamics of our proposed general graph structure can similarly be formulated and managed by solving a coupled system of equations.

Conceptually, DEQ focuses on weight-tied layers across an infinite depth. Viewing an infinite chain as a cycle, the core computational graph of DEQ can be considered a cyclic structure. Our work provides an alternative perspective: the fixed point found by a DEQ corresponds to a stable solution within a synchronous computational module defined by our framework. Our neural module formulation not only helps interpret this fixed point but also facilitates the analysis of the underlying implicit functions.

Furthermore, the architectural constraint known as the ”Universality of Single-Layer DEQ” implies that stacking multiple implicit layers is equivalent to a single implicit layer, which inherently limits the representational flexibility of deep implicit stacks. Scaling DEQs to models with a larger effective number of distinct computational nodes also remains a practical challenge noted in the literature.

In contrast, our framework generalizes the concept of an implicit layer. By moving from an infinite chain to a general graph structure, we propose a more flexible way to organize neural units, aiming to improve parameter efficiency and model performance. To address scalability, we introduce Neural Module (NM) regularization, which encourages the formation of decoupled subgraphs or modules. This provides a structured approach to manage multiple, potentially complex, computational subgraphs—an aspect less explicitly addressed in the standard DEQ formulation. Our analysis of NM regularization includes both theoretical justification and empirical studies on parameter selection, facilitating the handling of larger graph-based networks.

Empirically, the balanced and relatively independent neural modules formed under our framework appear to offer advantages in terms of efficiency and task performance, as shown in our experiments. These modules may help reduce overfitting (potentially enhancing generalization) and allow for parallel computation (improving efficiency). This addresses a consideration not central to the DEQ approach, which does not prescribe a specific strategy for modularizing computation to optimize these metrics.

C.1 DEEPER INSIGHTS INTO DEQ

While promising, certain theoretical aspects of DEQs warrant further exploration. In particular, more in-depth analysis could be beneficial for clarifying: (1) the representational role and significance of the fixed point beyond being a computational endpoint, and (2) the precise conditions under which weight-tied infinite-depth networks converge to such fixed points. Investigating these questions may further elucidate the behavior and advantages of networks with cyclic computational dependencies.

In cyclic neural structures, units are interdependent. Information propagates iteratively through the network until an equilibrium state is reached, which is then used for downstream tasks. This equilibrium is mathematically the fixed point of the system’s transformation.

The convergence to an equilibrium emerges from the mutual dependencies between neuron units, which form a system of equations that must be jointly satisfied. During computation in a cyclic module, each unit’s state is updated based on the states of others in the module. This process iterates until the updates become negligible, signaling convergence to a stable fixed point where the system's dynamics are in balance.

This iterative process resembles numerical methods for solving equation systems, such as fixed-point iteration or the Newton-Raphson method. Analogously, the cyclic network iteratively refines neuronal activations until a self-consistent solution (the fixed point) satisfying the internal relationships is found.

In summary, the equilibrium in cyclic neural networks results from an iterative process that seeks a stable solution to the system of equations defined by the interconnected units.

D Connection to Cyclic Structures in the Forward-Forward Algorithm

A key mechanism in cyclic Forward-Forward (FF) structures involves propagating activations through a cycle for a fixed number of steps. This can be seen as a finite-step numerical approximation to the solution of the equilibrium condition defined by the cycle's connections. While efficient, this approach may not guarantee convergence to a deterministic solution under all conditions, as it truncates the iterative process.

Moreover, cyclic FF structures typically operate on a predefined, fixed graph topology designed to align with the local learning rules of the FF algorithm, which favors certain architectural constraints.

In contrast, our framework explores the space of possible graph structures, effectively searching over a complete graph where any neuron may connect to any other. This offers greater flexibility but increases the complexity of identifying effective architectures. Therefore, our focus is on developing methods to efficiently discover performant subgraphs within this expansive space.

D.1 Key Considerations for Cyclic FF Structures and Proposed Alternatives

Several notable characteristics of cyclic FF structures motivate our design choices:

A. Architecture and Learning: They often rely on a predefined graph structure and employ local, layer-wise loss functions (like cross-entropy). This can limit architectural flexibility and may introduce an inductive bias based on the chosen preset topology.

B. Solution Method: They approximate the implicit system solution using a fixed, limited number of iteration steps, which may not always yield a consistent approximation.

C. Scale Management: The size of cyclic components is typically not explicitly regulated, which can affect computational complexity and learning dynamics.

Our framework addresses these aspects differently:

A. Our network topology is learned adaptively during training, allowing connections to form dynamically based on the task.

B. For identified cyclic components, we aim to solve the governing equations directly or ensure reliable convergence, moving beyond fixed-iteration approximations.

C. We employ NM regularization to influence the scale and decoupling of neural modules, facilitating parallel computation and managing complexity.

E Connection with OptNet

Earlier work on OptNet integrated optimization problems, specifically quadratic programs (QPs), as layers within neural networks, focusing on interactions between nodes. This design introduces a strong, specific inductive bias through the QP formulation. Furthermore, OptNet's backpropagation requires gradients from the QP solution, which depends on the preceding layer's output, potentially complicating the application and tuning of standard regularization techniques. These factors can make fine-tuning OptNet models for optimal performance and efficiency a challenging task.

Compared to this, the neural modules in our framework are based on general nonlinear transformations. They offer substantial flexibility for function approximation, as supported by universal approximation theorems for graph networks, without being constrained to a specific optimization-based structure. This allows for more straightforward control over model complexity and capacity, enabling a more direct balance between performance and efficiency—addressing some of the core challenges associated with optimization-based layers.

F ALGORITHM

We summarize our framework in three core algorithms: Algorithm 1 (Propagation), Algorithm 2 (Training Process), and Algorithm 3 (Prediction Process).

G THE SUPPLEMENTARY MATERIALS FOR EXPERIMENTS

This section provides detailed information about the experimental setup and analyses of NM regularization.

G.1 EXPERIMENTAL ENVIRONMENT

All experiments were conducted on the hardware platform specified in Table 2.

Table 2: Experimental Environment

CPU	Gen Intel(R) Core(TM) i9-12900H 2.90 GHz
Cores	16
Memory	32G
GPU	NVidia GeForce RTX 3060
Graphics Memory	12G

The parameter settings for the methods used in our experiments are described as Table 3.

Table 3: Parameter List

Mothods	**Parameters**
NN	learningRate optimized with the scale; momentum = 0.5; scaling learningRate = 1; weightPenaltyL2 = 0;
DEQ	learningRate optimized with the scale; momentum = 0.5; scaling learningRate = 0.1; weightPenaltyL2 = 0;
DAG	learningRate optimized with the scale; momentum = 0.5; scaling learningRate = 0.1; weightPenaltyL2 = 0;
OPTNET	learningRate optimized with the scale; momentum = 0.5; scaling learningRate = 0.1; weightPenaltyL2 = 0;
NM	learningRate optimized with the scale; momentum = 0.5; scaling learningRate = 1; weightPenaltyL2 = 0;
NM+L2	learningRate optimized with the scale; momentum = 0.5; scaling learningRate = 1; weightPenaltyL2 = 0; alpha = 0.001;
NM+NMS	learningRate optimized with the scale; momentum = 0.5; scaling learningRate = 1; weightPenaltyL2 = 0; alpha = 0.001;

G.2 COMPARISON WITH GNNS ON FACEBOOK DATA

As shown in Figure 8a, our Neural Modules (NMs), compared with the Modern Structure GNN, also achieved a significant performance gain on the Facebook dataset. This experiment further validates the effectiveness of our approach.

Algorithm 1 Propagation

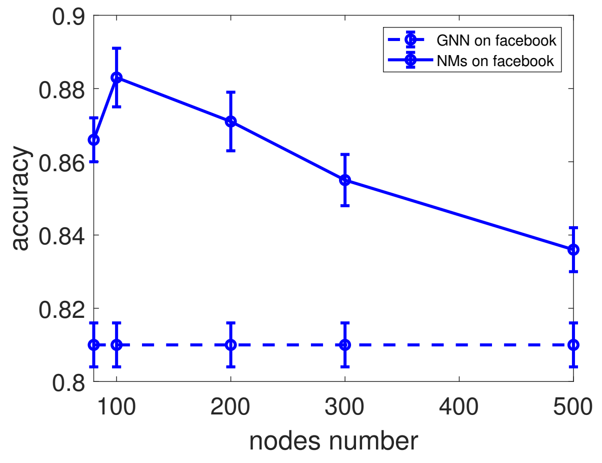
```
1: Input: Input vector $X^{in}$; Coefficient matrix $\mathcal{C}$; Input layer weights $W^1$; Output layer weights
   $W^m$; Propagation type (Forward/Backward) $PT$;
2: Output: Updated node value vector $X$
3: if $PT$ == Forward then
4:     Initialize $X = X^{in} \cdot W^{1T}$.
5: end if
6: if $PT$ == Backward then
7:     Initialize $X = X^{in} \cdot W^m$.
8: end if
9: Use Tarjan's algorithm on $\mathcal{C}$ to identify neural modules $\{$NMs$\}$.
10: Initialize $Node\_Queue = \{n_i | InArc(n_i) = 0\}$.
11: Initialize $NM\_Queue = \{NM_i | InArc(NM_i) = 0\}$.
12: while $Node\_Queue$ is not null and $NM\_Queue$ is not null do
13:     for $n_i$ in $Node\_Queue$ do
14:         if $PT$ == Forward then
15:             Execute forward propagation (as in traditional NNs).
16:         end if
17:         if $PT$ == Backward then
18:             Execute backward propagation (as in traditional NNs).
19:         end if
20:         Delete outgoing edges of $n_i$ from $\mathcal{C}$ .
21:     end for
22:     if $PT$ == Forward then
23:         Solve equations (1) and (9) for $NM_i \in NM\_Queue$ in parallel.
24:     end if
25:     if $PT$ == Backward then
26:         Solve equations (2) for $NM_i \in NM\_Queue$ in parallel.
27:     end if
28:     Delete edges associated with $NM_i \in NM\_Queue$ from $\mathcal{C}$.
29:     for $n_i$ associated with $\mathcal{C}$ do
30:         if $InArc(n_i)$ == 0 and $OutArc(n_i)$ != 0 then
31:             Enqueue $n_i$ to $Node\_Queue$.
32:         end if
33:     end for
34:     for $NM_i$ associated with $\mathcal{C}$ do
35:         if $InArc(NM_i)$ == 0 and $NM_i$ has edges then
36:             Enqueue $NM_i$ to$NM\_Queue$.
37:         end if
38:     end for
39: end while
40: if $PT$ == Forward then
41:     $X = f(X)$ (apply activation function $f$).
42: end if
43: if $PT$ == Backward then
44:     $X = f'(X)$ (apply derivative of activation function $f'$).
45: end if
46: return $X$
```

Algorithm 2 Training Process

```
1: Input: Initial input vector $X^0$; Ground-truth label vector $Y$; Initial adjacency matrix for the complete graph $\mathcal{K}$;Hyperparameter for thresholding (used to update $\gamma$) $k$; Input layer weights $W^1$; Output layer weights $W^m$;
2: Output: $\mathcal{K}$;$W^m$;$W_1$;$\gamma$
3: Initiate $\mathcal{K}$ as a random matrix (for the complete graph).
4: while the model has not converged do:
5:     Approximate $\mathcal{K}$ usingh hyperparameter $k$ to obtain the coefficient matrix $\mathcal{C}$.
6:     Run Propagation with $X^0$,$\mathcal{C}$,$W^1$,$W^m$,Forward.
7:     Compute predicted labels: $\widetilde{Y} = X \cdot W^{mT}$.
8:     Compute loss gradient $\nabla Y$.
9:     Run Propagation with $\nabla Y$,$\mathcal{C}^T$,$W^1$,$W^m$,Backward.
10:    Update $\mathcal{K}$ using equation (4).
11:    Update $W^m$ using equation (5).
12:    Update $W_1$ using equation (6).
13:    Update $\gamma$ to the $k$th largest absolute value in $\mathcal{K}$ (for thresholding).
14: end while
```

Algorithm 3 Predicting Process

```
1: Input: Input vector for prediction $X^0$; Threshold (from training) $\gamma$; Optimized adjacency matrix (from training) $\mathcal{K}$; Input layer weights $W^1$; Output layer weights $W^m$ ;
2: Output: Predicted label vector $\widetilde{Y}$.
3: Approximate $\mathcal{K}$ using $\gamma$ to obtain the coefficient matrix $\mathcal{C}$.
4: Run Propagation with $X^0$,$\mathcal{C}$,$W^1$,$W^m$,Forward.
5: Compute $\widetilde{Y} = X \cdot W^{mT}$.
6: return $\widetilde{Y}$
```

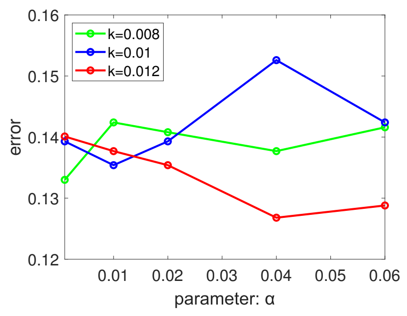


(a) Performance on Facebook Data

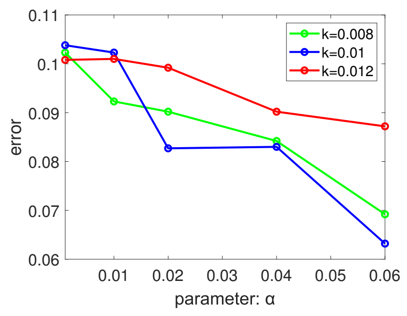
Figure 8: Performance gain achieved when our Neural Modules (NMs) were compared with a Modern Structure GNN on the Facebook dataset, which substantiates the broad applicability and effectiveness of our approach.

G.3 Optimization of Neural Modules via NM Regularization

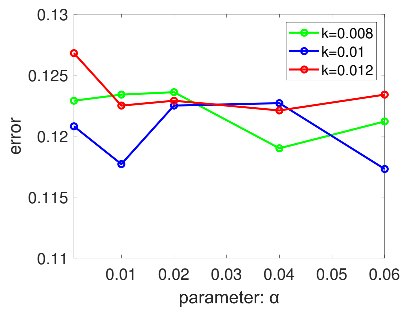
As discussed earlier, NM regularization yields significant improvements in model performance and efficiency. Here, we analyze parameter-tuning strategies for NM regularization and their relationship to dataset complexity.



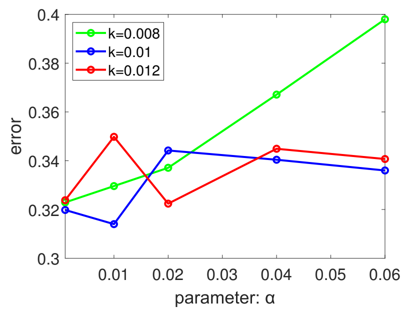
(a) Codon Usage Dataset



(b) Gas Dataset



(c) Facebook Dataset



(d) Activity Dataset

Figure 9: Quantitative Comparison of Topological Connectivity. Sparse connectivity metrics (e.g., edge density, average degree) across datasets, showing alignment with biological neural networks.

A key insight is that optimal parameter settings for NM regularization depend on dataset complexity:

Gas Dataset: Less complex; benefits from stronger regularization (reduces overfitting to simple patterns).

Activity Dataset: More complex; performs better with weaker regularization (preserves fine-grained patterns).

Codon Usage Dataset: Requires larger neural modules combined with stronger regularization (balances complexity and overfitting).

Facebook Dataset: Robust to parameter variations (exhibits stable performance across a range of regularization strengths).

Figure 9 visualizes these trends, showing how prediction error varies across datasets with the two key parameters α and k. The parameter k specifies the number of top edges retained, functioning similarly to the threshold parameter γ in its effect.

H The Effect of NM Regularization

Experimental results demonstrate three key benefits of NM regularization:

Captures Weight Trends: NM regularization accurately tracks weight evolution across iterations, enabling precise weight regularization.

Forms Balanced Modules: It groups nodes into independent, balanced neural modules—reducing redundancy and improving representational efficiency.

Boosts Parallel Efficiency: Since modules are independent, they can be processed in parallel. The overall efficiency of the model is determined by the size of the largest module (smaller, balanced modules minimize parallel bottlenecks).

H.1 Quantitative Analysis of NM Regularization

To formalize the effect of NM regularization, we define:

θ: Probability that an edge's weight is less than γ (the regularization threshold) in the current iteration.

q: Number of nodes in a neural module.

The probability that a new edge integrates into a module is given by:

$$p_{merge} = 1 - \theta^q \tag{16}$$

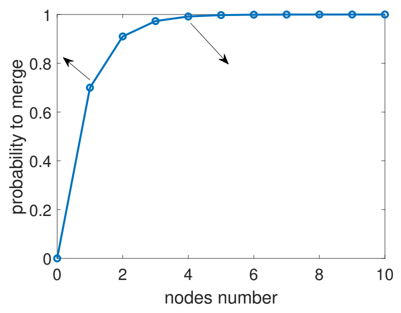
.

NM regularization dynamically adjusts this probability to balance module sizes:

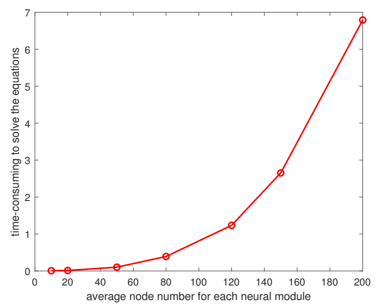
For small q (small modules), it increases p_{merge} to encourage module growth.

For large q (large modules), it decreases p_{merge} to prevent excessive expansion.

This balance minimizes disparities in merge probabilities across modules of varying sizes—ensuring the formation of compact, efficient modules (see Figure 10a).



(a) Merge probability of nodes' number per module.



(b) Time consumption on average module size.

Figure 10: (a) Merge probability of nodes' number per module (shows balanced growth); (b) Time consumption on average module size (demonstrates parallel efficiency—smaller, balanced modules reduce computation time).

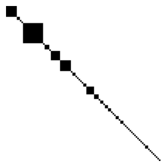
Additionally, Figure 10b illustrates the efficiency gains from parallel computation: as module sizes decrease (and balance improves), computation time decreases linearly—confirming that NM regularization enables effective scaling to large node counts.

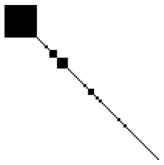
Figure11 show examples of automatically generated neural module structures for each dataset (black squares denote independent modules). These figures confirm that NM regularization produces sparse, balanced structures that mirror the connectivity patterns of biological neural networks (as quantitatively verified in Figure 12)

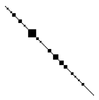
Figure 11 further quantifies this by comparing topological connectivity patterns across datasets—confirming that our framework consistently generates structures aligned with biological neural networks.

I The Proof of the Universal Approximation

In the forward propagation process, the system of equations defined by our general graph structure constructs implicit functions across all nodes in a neural module. For any node n_i this implicit function can be transformed into an explicit function of the input X^0.


(a) Codon Usage Dataset


(b) Gas Dataset


(c) Facebook Dataset

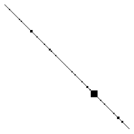
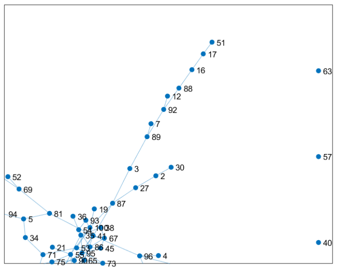
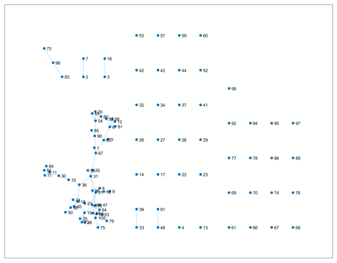
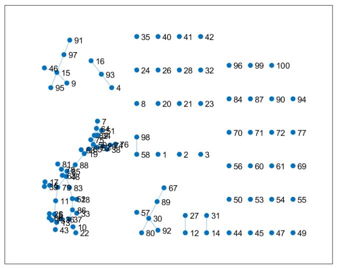

(d) Activity Dataset

Figure 11: Automatically Formed Neural Modules (Codon Usage, Gas, Facebook, and Activity Datasets). Each black square represents an independent module.

‘


(a) Codon Usage Dataset


(b) Gas Dataset


(c) Facebook Dataset

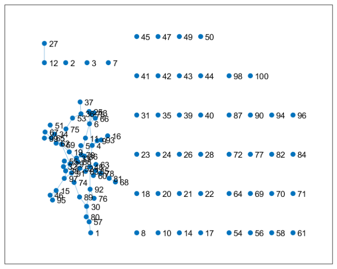

(d) Activity Dataset

Figure 12: Connectivity Patterns from NM Regularization. Upper triangular adjacency matrices (simplified for visualization) show sparse, biologically plausible connectivity.

‘

By the Universal Approximation Theorem (Hornik et al., 1989), any continuous function $f : \mathcal{X} \to \mathbb{R}$ (where $\mathcal{X} \in \mathbb{R}^d$ is compact) can be approximated to arbitrary precision by a single hidden-layer feed-forward network with a non-linear activation function.

Our framework satisfies this theorem's conditions:

The system of equations includes a non-linear activation function (applied to node values in forward propagation).

The general graph structure enables arbitrary connections between nodes—effectively acting as a feed-forward network with flexible layer definitions.

J THE PROOF OF THEOREM 3.1

Proof. The gradient of the loss with with respect to edge w_{ij} under NM regularization is given by

$$\frac{\partial J_{NM}}{\partial w_{ij}} = \frac{\partial J}{\partial w_{ij}} + \alpha r_{ij} w_{ij}, \tag{17}$$

Using stochastic gradient descent (SGD) for weight updates, the update rule becomes:

$$w_{ij} \leftarrow w_{ij} - \eta(\frac{\partial J}{\partial w_{ij}} + \alpha r_{ij} w_{ij}), \tag{18}$$

where η is the learning rate. □

K THE PROOF OF THEOREM 3.2

Proof. Assume the i-th column of $\mathcal{C}$ (weights from all nodes to node i) follows a multivariate normal prior:

$$w_{:i} \sim \mathcal{N}(0, \tau^2 diag(r_{:i})^{-1}), \tag{19}$$

where τ^2 is the prior variance, and $r_{:i}$ is the vector of regularization coefficients for the i-th column.

Consider a Bayesian linear regression model for node $y = Xw_{:i} + \epsilon$ and $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$. The posterior distribution of $w_{:i}$ substituts the Gaussian likelihood and prior::

$$\begin{aligned} p(w_{:i}|y, X) &\propto p(y|X, w_{:i})p(w_{:i}) \\ &\propto exp(-\frac{\|y - Xw_{:i}\|^2}{2\sigma^2})exp(-\frac{\|w_{:i}^T \cdot diag(r_{:i}) \cdot w_{:i}\|^2}{2\tau^2}) \end{aligned}$$

Taking the negative logarithm (to convert to a minimization problem), the maximum a posteriori (MAP) estimate of

$$\arg\min_{w_{:i}}\{\frac{1}{2\delta^2}\|y - Xw_{:i}\|^2 + \sum \frac{r_{ji}}{2\tau^2}\|w_{ji}\|^2\}, \tag{20}$$

which equivalent to NM regularization as

$$\arg\min_{w_{:i}}\{\|y - Xw_{:i}\|^2 + \alpha \sum r_{ji}\|w_{ji}\|^2\}. \tag{21}$$

This is exactly the NM regularization objective. Thus, NM regularization corresponds to a Bayesian prior on edge weights.

To confirm the posterior distribution, expand the log-posterior:

$$\begin{aligned} logp(w_{:i}|y, X) &\propto -\frac{1}{2\delta^2}\|y - Xw_{:i}\|^2 - \frac{1}{2\tau^2}w_{:i}^T diag(r_{:i})w_{:i} \\ &\propto -\frac{1}{2\delta^2}(w_{:i}^T X^T X w_{:i} - 2w_{:i}^T X^T y) - \frac{1}{2\tau^2}w_{:i}^T diag(r_{:i})w_{:i} \\ &\propto -\frac{1}{2}w_{:i}^T(\frac{X^T X}{\delta^2} + \frac{1}{\tau^2}diag(r_{:i}))w_{:i} + \frac{w_{:i}^T X^T y}{\delta^2} \end{aligned}$$

This matches the norm of a multivariate normal distribution's log-pdf. By converting the distribution to a normalized form, Theorem 3.3 holds. □

L The Proof of Theorem 3.3

Proof. By Theorem 3.4, edge weights follow a normal distribution. When pruning edges based on the absolute value of weights (using threshold γ), the weights in each iteration follow a **folded normal distribution**. For simplicity, consider the cumulative distribution function (CDF) of the standard folded normal distribution:

$$P(w) = 2\Phi(\frac{w}{\Delta}) - 1. \tag{22}$$

where Φ is the CDF of the standard normal distribution, and Δ is the scale parameter. NM regularization reduces Δ over iterations (by shrinking weights toward zero via the $\alpha * r_{ij} * w_{ij}$. This causes the folded normal distribution to concentrate around zero, meaning learned parameters tend to approach zero with the decrease in Δ.

For any iteration t, let w_t be the parameters of the original model and $\tilde{w_t}$ be the parameters of the optimized model by k. The shrinkage effect of NM regularization implies:

$$\|w_t - \tilde{w_t}\|^2 < \|w_0 - \tilde{w_0}\|^2. \tag{23}$$

where w_0 is the initial weight.

Follow (Lin et al., 2020), let $\zeta = \frac{c}{\sqrt{T}}$, $c = \sqrt{\frac{f(w_0)-f(w_*)}{LG^2}}$ and T be the number of iteration.

$$\begin{aligned}\mathbb{E}\|\nabla u\|^2 &\leq \frac{f(w_0) - f(w^*)}{\zeta(T+1)} + L\zeta G^2 + \frac{L^2}{T+1}\sum_{t=0}^{T}\mathbb{E}\|w_t - \tilde{w_t}\|^2 \\ &\leq \frac{2(f(w_0) - f(w^*))}{\zeta(T+1)} + L\zeta G^2 + L^2\|w_0 - \tilde{w_0}\|^2\end{aligned}$$

□

M The Usage of Large Language Models (LLMs) in This Paper

In this paper, Large Language Models (LLMs) were only utilized in three specific aspects, with their application scope strictly limited to auxiliary text and code polishing:

1. Correcting grammatical errors in the manuscript;

2. Rectifying typos throughout the text;

3. Fixing minor code errors in the experimental section.

Notably, LLMs were *not* employed for core academic work, including the generation of key research ideas, the derivation of critical mathematical proofs, or any other tasks that involve original academic reasoning.