AUTOMATED ARCHITECTURE SYNTHESIS FOR ARBITRARILY STRUCTURED NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper presents a novel perspective on the architecture of Artificial Neural Networks (ANNs). Conventional ANNs typically adopt predefined tree-like or Directed Acyclic Graph (DAG) structures for simplicity; however, these structures restrict network collaboration and capability due to the absence of horizontal and backward communication. In contrast, biological neural systems comprise billions of neural units with highly complex connection patterns, enabling each neuron to establish connections with others based on specific contextual requirements. Inspired by biological neural systems, this study proposes a new framework that automatically learns to construct arbitrary graph structures during the training process. It also introduces the concept of "Neural Modules" to organize neural units, which facilitates communication between any nodes. Unlike traditional ANNs that rely on DAGs, the proposed framework evolves from complete graphs, allowing unrestricted communication between neurons—mimicking the operational behavior of biological neural networks. Furthermore, we develop a method to compute these arbitrary graph structures and a regularization technique to organize them into multiple independent, balanced Neural Modules. This approach reduces overfitting and enhances efficiency through parallel computing. Overall, our method enables ANNs to learn effective arbitrary structures analogous to those of biological neural systems. It exhibits strong adaptability to various tasks and compatibility across different scenarios, with experimental results validating its potential.

1 INTRODUCTION

This work offers a fresh viewpoint on the architecture of Artificial Neural Networks (ANNs). Traditional ANNs are often hierarchically arranged in tree-like structures or DAGs, either through predefined designs or via Neural Architecture Search (NAS) methods that explore within a DAG space. However, this conventional approach impedes effective communication among nodes and introduces significant structural bias. In practice, existing ANN structures are optimized within a limited search space, which constrains the potential of ANNs and prevents them from achieving their full capabilities.

Our work rethinks traditional ANNs by arguing that current connectivity approaches fail to capture the true nature of neural networks. Nodes in an asynchronous tree-like structure cannot establish flexible connections, which hinders information transfer between neural units and leads to inherent deficiencies. To address this limitation, we propose a method to construct a synchronous graph structure for nodes using the introduced Neural Modules, thereby promoting collaboration among neural units.

In fact, some prior studies have identified the issues with current ANN structures and attempted to design cyclic graph structures for ANNs. Nevertheless, these works neither analyzed the essence of generalized cyclic structures for ANNs nor proposed a framework to automatically formulate such structures—analogous to biological neural networks. These gaps are precisely the academic focus of this paper.

Our approach enables synchronous communication among all nodes within the structure and introduces a method for dynamically forming these structures during the learning process. These

enhancements improve information transfer efficiency, thereby boosting the overall capacity of neural network (NN) architectures. By fostering collaboration among nodes and enabling automatic structure learning, our method harnesses the collective power of neural networks, unlocking their potential in a manner similar to biological neural networks.

It is important to note that conventional tree-like neural networks are essentially a subset of our designed general graph structure. We clarify the inherent bias in current neural network architectures and provide an analysis of the integration of general graph structures into neural networks in the appendix. Within our framework, multiple neural units collaboratively and automatically execute precise functional implementations during the learning process. Our innovation aims to bridge the gap between ANNs and more generalized structures that resemble biological neural networks.

Designing this architecture poses significant challenges, as the framework introduces higher computational demands and an increased risk of overfitting. To address these issues, we propose a novel regularization method. This method efficiently organizes nodes into multiple independent neural modules that can be processed in parallel on modern GPUs. It enables automatic node organization, thereby enhancing learning efficiency, reducing overfitting, and ultimately improving overall performance.

Our learning process demonstrates strong adaptability to diverse tasks. We evaluated our optimization method with state-of-the-art networks, and the results show that it outperforms them in many real-world scenarios.

In summary, the key contributions of this study are:

1. We analyze the bias in existing tree-like neural network structures and provide a detailed analysis of our proposed architectural improvements.

2. We develop a method for ANNs to automatically learn and construct arbitrary graph structures.

3. We introduce a novel regularization technique that organizes neural units into the proposed Neural Modules, thereby enhancing structural efficiency through parallel computing and improving performance by reducing overfitting.

2 Related Works

To advance the existing tree-like structure for NNs, Yuan (Kun Yuan & Yan, 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 & Sun, 2016) (Huang & Weinberger, 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 (Liang Wei Yang & ilip S. Yu, 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 (Liang Wei Yang & ilip S. Yu, 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, (Liang Wei Yang & ilip S. Yu, 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, (Liang Wei Yang & ilip S. Yu, 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 & Liu, 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.

Our work also involves NN compression. In recent years, various compression algorithms have been developed, including quantization (Kai Han & Xu, 2020) (Mingzhu Shen & Wang) (Yang He & Yang, 2018), low-rank approximation (Li & Shi, 2018) (Zhaohui Yang & Xu, 2019) (Xiyu Yu & Tao, 2017), knowledge distillation (Shumin Kong & Xu, 2020) (Shan You & Tao, 2018), and network pruning (Pavlo Molchanov & Kautz., 2019). In this paper, we seek to improve weight pruning for our framework using a method similar to (Tao Lin, 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.

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

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 (Schrauwen & Campenhout., 2007) (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 (Esteban Real & Kurakin, 2017) (Zoph & Le, 2017) (Hieu Pham & Quoc V. Le, 2018) (Mingxing Tan & Le, 2019) (Arber Zela & Hutter, 2020) (Joe Mellor & Crowley, 2021). Modern methods like DARTS (Hanxiao Liu & Yang, 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. 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 structure is generated by searching within a set of complete graphs and selecting the top k edges using pruning techniques in each iteration. Unlike conventional NAS methods that search within a DAG, our approach can generate arbitrary graph structures.

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:

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

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. For the elements in $\widetilde{\mathcal{G}}$, we introduce the following theorem:

Theorem 3.1 (Universal Graph Approximation Theorem)**.** *Let $\mathcal{F}$ be an implicit function defined on a compact set, which can be transformed into a continuous explicit function for all variables. In this case, there exists a graph that can effectively approximate the function $\mathcal{F}$.*

3.4 Backward Process

To compute 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:

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

Finally, we can compute the gradient of the edges. First, for the gradient of the edges in the complete graph $\mathcal{G}$, according to the system of equations, we need to consider the gradient of each node. For

any jth node in the graph, the weights of the incoming edges are represented by the jth($1 \leq j \leq p$) of the complete graph $\mathcal{G}$. For convenience, we introduce the following operator:

$$\mathcal{H}_j = [f(x_1), ..., f(x_{j-1}), 1, f(x_{j+1}), ..., f(x_p)] \,, \tag{3}$$

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

$$\nabla W_{:j}^T = \nabla x_j \circ f'(x_j) \cdot \mathcal{H}_j \,. \tag{4}$$

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

$$\nabla W^m = \nabla X^{mT} \circ g'(X^{mT}) \cdot f(X) \,. \tag{5}$$

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

$$\nabla W^1 = \nabla X^T \circ f'(X^T) \cdot X^0 \,. \tag{6}$$

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

3.5 Neural Module Optimization

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 :

$$d_{ij} = e^{-\widetilde{w}_{ij}}. \tag{7}$$

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

Third, 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

$$r_{ij} = \frac{z_i * z_j}{d_{ij}} \tag{8}$$

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:

$$J_{NM}(x_{:i}) = J(x_{:i}) + \alpha \sum_{j=1}^{p} r_{ji} \cdot w_{ji}^2, \tag{9}$$

where J denotes the objective function.

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

Theorem 3.2. *For NM regularization, in each iteration with learning rate η, parameter w_{ij} upgrades as follow:*

$$w_{ij} \leftarrow w_{ij}(1 - \eta\alpha r_{ij}) - \eta \frac{\partial J}{\partial w_{ij}} \tag{10}$$

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 3.3. *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:*

$$w_{:i} \sim \mathcal{N}((X^T X + \lambda diag(r_{:i}))^{-1} X^T y, \delta^2 (X^T X + \lambda diag(r_{:i}))^{-1})), \tag{11}$$

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.

Let w^* be the parameters of global optimal model, w_0 be the parameters of the original model and $\tilde{w_0}$ be the parameters of the optimized model by k 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 3.4. *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:*

$$\mathbb{E}\|\nabla u\|^2 = \mathcal{O}(\sqrt{\frac{L(f(w_0) - f(w^*))}{T}} G + L^2 \|w_0 - \tilde{w_0}\|^2). \tag{12}$$

3.6 ALGORITHM

Note that directly solving Equations 1 and 2 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 $\mathcal{C}$ using a parameter, 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 the approximated graph 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 the approximated graph 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.

In our algorithm, Tarjan's algorithm (Tarjan., 1976) is used to identify strongly connected components, which serve as the foundation for forming Neural Modules.

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. 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

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. The results are presented in tables 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.

4.1 PERFORMANCE EVALUATION OF NEURAL MODULES BEYOND TRADITIONAL TREE-LIKE STRUCTURES

We compared the performance of our framework with baselines that go beyond traditional tree-like structures (e.g., DEQ and OPTNET) 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.

4.2 PERFORMANCE OF NEURAL MODULES WITH FF AND DARTS

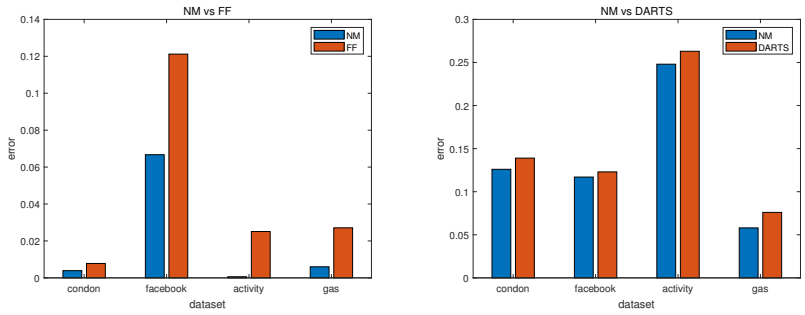
To validate our approach, we conducted comparative experiments using the Cyclic Forward-Forward algorithm (Liang Wei Yang & ilip S. Yu, 2024),by formulating the multi-class classification task as a binary classification task on the four datasets. We also compared our method with the well-known NAS method DARTS (Hanxiao Liu & Yang, 2019). As shown in Figure 1, our Neural Module (NM) framework achieves higher accuracy than both the Cyclic Architecture with the Forward-Forward Algorithm and Differential Architecture Search (DARTS).

4.3 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.

Table 1: The performance of algorithms

Methods	80	100	200	Condon 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	$\mathbf{0.158 \pm 0.006}$	$\mathbf{0.178 \pm 0.003}$
Methods	80	100	200	Activity 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&L1	$\mathbf{0.348 \pm 0.003}$	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	$\mathbf{0.287 \pm 0.003}$
Methods	80	100	200	Facebook 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&L1	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	$\mathbf{0.145 \pm 0.004}$	$\mathbf{0.164 \pm 0.006}$	$\mathbf{0.166 \pm 0.005}$	$\mathbf{0.162 \pm 0.006}$
Methods	80	100	200	Gas 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&L1	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	$\mathbf{0.132 \pm 0.003}$	$\mathbf{0.108 \pm 0.005}$

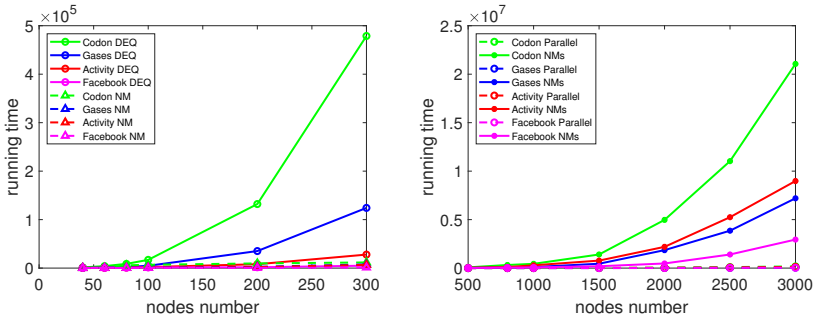


(a) Neural Modules vs FF (b) Neural Modules vs DARTS

Figure 1: It can be observed that our model achieves a lower error rate than both the Cyclic Forward-Forward algorithm and DARTS.

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 2a 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 2: The efficiency of NM regularization

In Figure 2b, 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. In Figure 2, the unit of measurement is the second.

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 evolving from tree-like structures to more general graph structures. Despite 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.

K. Ahmed and L. Torresani. Mask connect connectivity learning by gradient descent. In *ECCV*, 2018.

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

Tonmoy Saikia Yassine Marrakchi Thomas Brox Arber Zela, Thomas Elsken and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *NeurIPS*, 2019.

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

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

Shutong Ding, Tianyu Cui, Jingya Wang, and Ye Shi. Two sides of the same coin: Bridging deep equilibrium models and neural odes via homotopy continuation. *arXiv preprint arXiv:2310.09583*, 2023.

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

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. pp. 417–426, 2019.

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

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

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*, 2005.

Karen Simonyan Hanxiao Liu and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019.

Zhang X. Ren S. He, K. and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

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.

Barret Zoph Hieu Pham, Melody Guan and Jeff Dean Quoc V. Le. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Liu Z. VanDer Maaten L. Huang, G. and K.Q.. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

Amos Storkey Joe Mellor, Jack Turner and Elliot J Crowley. Neural architecture search without training. In *ICML*, 2021.

Yixing Xu Chunjing Xu Enhua Wu Kai Han, Yunhe Wang and Chang Xu. Training binary neural networks through learning with noisy supervision. In *ICML*, 2020.

Jing Shao Kun Yuan, Quanquan Li and Junjie Yan. Learning connectivity of neural networks from a topological perspective. In *ECCV*, 2020.

Chong Li and CJ Richard Shi. Constrained optimization based low-rank approximation of deep neural networks. In *ECCV*, 2018.

Weizhi Zhang Zihe Song Jing Ma Jiawei ZhangPh Liang Wei Yang, Hengrui Zhang and ilip S. Yu. Beyond directed acyclic computation graph with cyclic neural network. In *Arxiv*, 2024.

Zenan Ling, Zhenyu Liao, and Robert C Qiu. On the equivalence between implicit and explicit neural networks: A high-dimensional viewpoint. *arXiv preprint arXiv:2308.16425*, 2023.

Jiaming Liu, Xiaojian Xu, Weijie Gan, Ulugbek Kamilov, et al. Online deep equilibrium learning for regularization by denoising. *Advances in Neural Information Processing Systems*, 35:25363–25376, 2022.

Ruoming Pang Vijay Vasudevan Mark Sandler Andrew Howard Mingxing Tan, Bo Chen and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

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

Stephen Tyree Iuri Frosio Pavlo Molchanov, Arun Mallya and Jan Kautz. Importance estimation for neural network pruning. In *CVPR*, 2019.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 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.

David Verstraeten Benjamin Schrauwen and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *In In Proceedings of the 15th european symposium on artiffcial neural networks. p. 471-482 2007, pages 471–482*, 2007.

Chao Xu Shan You, Chang Xu and Dacheng Tao. Learning with single-teacher multi-student. In *AAAI*, 2018.

Shan You Shumin Kong, Tianyu Guo and Chang Xu. Learning student networks with few data. In *AAAI*, 2020.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *Computer Science*, 2015.

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

Luis Barba Daniil Dmitriev Martin Jaggi Tao Lin, Sebastian U. Stich. Dynamic model pruning with feedback. In *ICLR*, 2020.

Robert E. Tarjan. Depth-first search and linear graph algorithms. In *In SIAM Journal on Computing 1 (1972), 146–160.*, 1976.

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

Xinchao Wang Xiyu Yu, Tongliang Liu and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *CVPR*, 2017.

Yan J. Luo C. Chen L. Lin Q. Yan, K. and D. Zhang. A surrogate objective framework for prediction+optimization with soft constraints. 2021.

Li P. Pang T. Yang, Z. and Y. Liu. Improving adversarial robustness of deep equilibrium models with explicit regulations along the neural dynamics. In *ICML*, 2023.

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

Xuanyi Dong Yanwei Fu Yang He, Guoliang Kang and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018.

Sweah Liang Yong, Markus Hagenbuchner, Ah Chung Tsoi, Franco Scarselli, and Marco Gori. Document mining using graph neural network. In *Comparative Evaluation of XML Information Retrieval Systems: 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006, Dagstuhl Castle, Germany, December 17-20, 2006, Revised and Selected Papers 5*, pp. 458–472. Springer, 2007.

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

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

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *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:

$$\begin{cases} X^0 \cdot W^{1T} = X^1 \\ X^1 \cdot W^{2T} = X^2 \\ ... \\ X^{m-1} \cdot W^{mT} = X^m. \end{cases} \tag{13}$$

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 3.

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

Figure 3: 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.

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

Figure 4: The Coefficient Matrix $\mathcal{C}$ for DAG(e.g. Resnet)

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 4.

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

However, the lower triangular coefficient matrix still represents the limit of asynchronous structures, with the potential of the upper triangular region remaining largely untapped. In our work, we extend asynchronous structures to synchronous ones, breaking the constraints of lower triangular matrices and generalizing the coefficient matrix $\mathcal{C}$ to a full adjacency matrix of a general graph. This generalized structure enables a more comprehensive and interconnected network, delivering greater flexibility and performance. Ultimately, our framework transitions neural network architectures from tree-based structures to general graph structures by expanding the coefficient matrix to span all levels.

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 5.

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

Figure 5: 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 THE CONNECTION WITH DEEP EQUILIBRIUM MODELS (DEQ)

Previous research has identified stable points in infinite-level neural networks with identical weights. Deep Equilibrium Models (DEQ) address this phenomenon by modeling the problem across infinite levels and simplifying it to an implicit function, where finding a fixed point is equivalent to solving for the root of this implicit function. In our work, we demonstrate that our general graph structure can also be managed by solving a system of functions.

In fact, DEQ focuses on infinite levels with identical weights; if we treat an infinite path as a cycle, the core research object of DEQ is essentially a cyclic graph. In this paper, we reveal the fundamental nature of fixed points: they are solutions within our synchronous neural module structure. Our neural modules not only help clarify this essence but also enable concurrent visualization of implicit functions.

Furthermore, the "Universality of Single-Layer DEQ" establishes that multiple implicit hidden layers are equivalent to a single implicit hidden layer—imposing a fundamental limitation on DEQ's ability to handle multiple implicit layers. Most critically, as noted in prior work, DEQ struggles to scale to larger numbers of nodes.

In contrast, our framework reinterprets implicit hidden layers—shifting from infinite levels to elegant general graphs—to organize neural units optimally, improving both model efficiency and performance. To address scalability, we introduce NM regularization, which groups nodes into independent neural modules. Unlike DEQ, our framework provides a principled methodology for handling multiple neural modules. Our analysis of NM regularization includes both theoretical foundations and experimental parameter tuning, enabling effective management of large-scale graphs.

Moreover, the independent, balanced neural modules generated by our framework outperform DEQ in both efficiency and performance (as demonstrated in experiments). These modules reduce overfitting (enhancing performance) and enable parallel computation (boosting efficiency)—addressing a key gap in DEQ, which does not discuss strategies for organizing neural units to optimize these metrics.

C.1 DEEPER INSIGHTS INTO DEQ

Despite its potential, the theoretical foundations of DEQ remain underexplored. There is a notable lack of in-depth analysis regarding two critical questions: (1) What is the significance of fixed points in DEQ? (2) Why do infinite sequences of identical weights lead to such fixed points? This gap presents an opportunity for further research to clarify the behavior of neural networks with cyclic connections and their advantages over traditional architectures.

In cyclic structures, neuronal units are interconnected and interdependent. Information flows recursively through multiple iterations until the system reaches an equilibrium state, which is then propagated to other units. This equilibrium state is, in essence, the fixed point of the implicit hidden layers.

The propagation of information toward equilibrium (a fixed point) in cyclic structures arises from the interdependence of neuron units. As outlined in our prior work, these units are not independent but form a complex system of equations that must be satisfied.

In a cyclic neural network, information iterates through the network, with each unit's state influenced by the states of other units in the same neural module. This process continues until changes between iterations become negligible—indicating convergence to an equilibrium state. At this fixed point, the system's dynamics stabilize, with no further significant changes in unit states.

This convergence to equilibrium can be analogized to numerical methods for solving systems of equations—such as the Newton-Raphson method, an iterative technique for finding successively better approximations to the roots of a real-valued function. Similarly, in cyclic neural networks, information flow iterates until it finds a stable solution that satisfies the system's equations—this solution is the fixed point.

In summary, the propagation toward equilibrium (a fixed point) in cyclic neural networks is the result of iteratively solving the system of equations formed by interdependent neuron units, where the system seeks a stable state that represents its equilibrium.

D THE CONNECTION TO CYCLIC STRUCTURES WITH THE FORWARD-FORWARD ALGORITHM

The key algorithm in cyclic FF structures involves propagating information through the cyclic graph for a predefined number of iterations—an attempt to numerically approximate the solution to the system of equations capturing the graph's interdependencies. This approach relies on a fixed number of iterative numerical steps (as analyzed earlier) but cannot guarantee deterministic outcomes under all operating conditions.

Additionally, cyclic FF structures use a predefined graph topology, which simplifies the interrelationships between nodes to suit the FF algorithm—enabling more efficient operation within the fixed architectural framework.

In contrast, our framework searches for optimal graph structures within a complete graph space. Unlike traditional layered architectures (where each unit may connect to any unit in adjacent layers), a complete graph allows every neuron to connect to every other neuron. While this flexibility expands the design space, it also introduces complexity in structure search. Thus, our framework focuses on developing methods to search for optimized structures within this complete graph space—enhancing both performance and efficiency.

Key Limitations of Cyclic FF Structures and Our Solutions

Cyclic FF structures suffer from three critical limitations, all of which are addressed by our framework:

Limitations of Cyclic FF Structures:

A. Relies on a predefined graph structure and uses cross-entropy as a local loss function for each unit and its neighbors (aligning with FF). This limits flexibility and relies on greedy local optimization and introduces bias by using predefined structure.

B. Approximates the solution to the system of equations using a fixed number of iterations—failing to guarantee deterministic results.

C. Does not account for cycle size, which critically impacts framework complexity, efficiency, and performance.

Our Solution:

A. Our network structure is learned (not predefined). During training, neurons dynamically form connections to generate an optimal topology.

B. For each cyclic component, we solve the system of equations directly to obtain an exact solution, eliminating the need for iterative approximation.

C. We use NM regularization to regulate the size of neural modules (and their cyclic components) while leveraging parallel computing to improve efficiency.

E THE CONNECTION WITH OPTNET

Earlier research on OptNet focused on modeling interconnections between nodes in the same level of traditional tree-like structures. OptNet implements quadratic programming (QP) within these nodes, introducing significant structural bias. Furthermore, OptNet uses parameters derived from QP problems for backpropagation—with these parameters determined by nodes in the previous layer—creating challenges for regularization (as discussed earlier). These flaws make it difficult to fine-tune OptNet's performance and overall efficiency.

In contrast, our neural modules are nonlinear and exhibit sufficient flexibility to approximate any compact function (as guaranteed by the Universal Graph Approximation Theorem) without introducing bias. Additionally, we can easily control parameter complexity to simultaneously enhance performance and efficiency—addressing OptNet's core limitations.

F ALGORITHM

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

Algorithm 1 Propagation

Input: Input vector X^{in}; Coefficient matrix $\mathcal{C}$; Input layer weights W^1; Output layer weights W^m; Propagation type (Forward/Backward) PT;
Output: Updated node value vector X
if PT == Forward **then**
    Initialize $X = X^{in} \cdot W^{1T}$.
end if
if PT == Backward **then**
    Initialize $X = X^{in} \cdot W^{m}$.
end if
Use Tarjan's algorithm on $\mathcal{C}$ to identify neural modules {NMs}.
Initialize $Node\_Queue = \{n_i | InArc(n_i) = 0\}$.
Initialize $NM\_Queue = \{NM_i | InArc(NM_i) = 0\}$.
while $Node\_Queue$ is not null and NM_Queue is not null **do**
    for n_i in $Node\_Queue$ **do**
        if PT == Forward **then**
            Execute forward propagation (as in traditional NNs).
        end if
        if PT == Backward **then**
            Execute backward propagation (as in traditional NNs).
        end if
        Delete outgoing edges of n_i from $\mathcal{C}$.
    end for
    if PT == Forward **then**
        Solve equations (1) and (9) for $NM_i \in NM\_Queue$ in parallel.
    end if
    if PT == Backward **then**
        Solve equations (2) for $NM_i \in NM\_Queue$ in parallel.
    end if
    Delete edges associated with $NM_i \in NM\_Queue$ from $\mathcal{C}$.
    for n_i associated with $\mathcal{C}$ **do**
        if $InArc(n_i)$ == 0 and $OutArc(n_i)$!= 0 **then**
            Enqueue n_i to $Node\_Queue$.
        end if
    end for
    for NM_i associated with $\mathcal{C}$ **do**
        if $InArc(NM_i)$ == 0 and NM_i has edges **then**
            Enqueue NM_i toNM_Queue.
        end if
    end for
end while
if PT == Forward **then**
    $X = f(X)$ (apply activation function f).
end if
if PT == Backward **then**
    $X = f'(X)$ (apply derivative of activation function f').
end if
return X

Algorithm 2 Training Process

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 γ) k; Input layer weights W^1; Output layer weights W^m;
Output: $\mathcal{K}$;W^m;W_1;γ
Initiate $\mathcal{K}$ as a random matrix (for the complete graph).
while the model has not converged **do**:
    Approximate $\mathcal{K}$ usingh hyperparameter k to obtain the coefficient matrix $\mathcal{C}$.
    Run Propagation with X^0,$\mathcal{C}$,W^1,W^m,Forward.
    Compute predicted labels: $\widetilde{Y} = X \cdot W^{mT}$.
    Compute loss gradient ∇Y.
    Run Propagation with ∇Y,$\mathcal{C}^T$,W^1,W^m,Backward.
    Update $\mathcal{K}$ using equation (4).
    Update W^m using equation (5).
    Update W_1 using equation (6).
    Update γ to the kth largest absolute value in $\mathcal{K}$ (for thresholding).
end while

Algorithm 3 Predicting Process

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

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 1.

Table 2: Experimental Environment

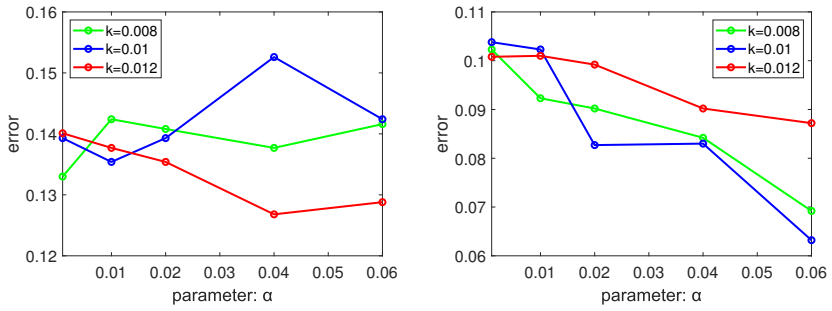
CPU	Gen Intel(R) Core(TM) i9-12900H 2.90 GHz
Cores	16
Memory	32G
GPU	NVidia GeForce RTX 3060
Graphics Memory	12G

G.2 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.

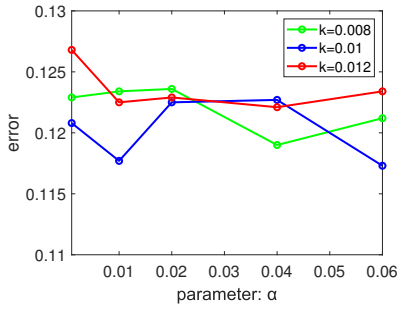
A key insight is that optimal parameter settings for NM regularization depend on dataset complexity:

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

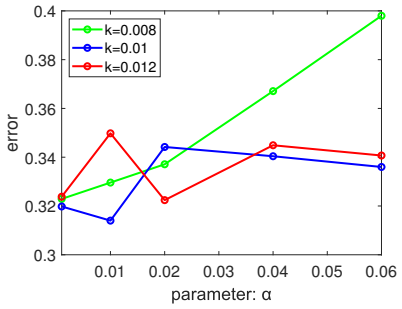


(a) Codon Usage Dataset

(b) Gas Dataset



(c) Facebook Dataset



(d) Activity Dataset

Figure 6: Quantitative Comparison of Topological Connectivity. Sparse connectivity metrics (e.g., edge density, average degree) across datasets, showing alignment with biological neural networks.

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).

These trends are visualized in Figure 6, which shows the impact of two key parameters (α and k) on prediction error across datasets.

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:

$$p_{merge} = 1 - \theta^q \tag{14}$$

.

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 7a).

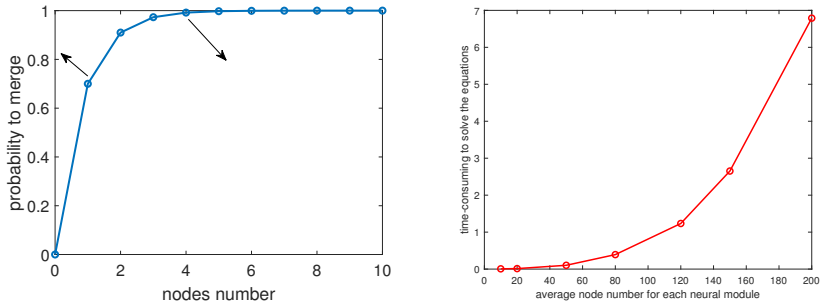


Figure 7: (a) Merge probability vs. number of nodes per module (shows balanced growth); (b) Time consumption vs. average module size (demonstrates parallel efficiency—smaller, balanced modules reduce computation time).

Additionally, Figure 7b 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.

Figure 8 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 9)

Figure 9 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 THEOREM 3.1

Proof. 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.

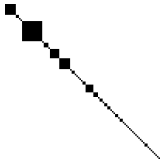
By the Universal Approximation Theorem (Hornik et al., 1989), any continuous function $f : \mathcal{X} \to \mathbb{R}$ (where $\mathcal{X} \in \mathbb{R}^d$ is compact) can be approximated to arbitrary precision by a single hidden-layer feed-forward network with a non-linear activation function.

Our framework satisfies this theorem's conditions:

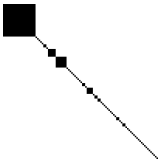
The system of equations includes a non-linear activation function (applied to node values in forward propagation).

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

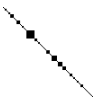
Thus, Theorem 1 holds. □



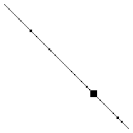
(a) Codon Usage Dataset



(b) Gas Dataset

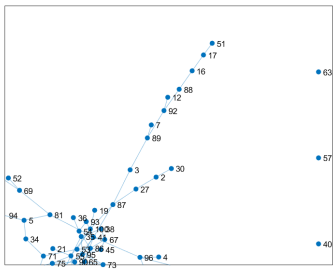


(c) Facebook Dataset

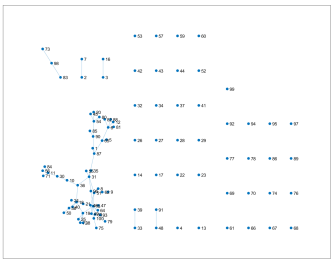


(d) Activity Dataset

Figure 8: Automatically Formed Neural Modules (Codon Usage, Gas, Facebook, and Activity Datasets). Each black square represents an independent module.

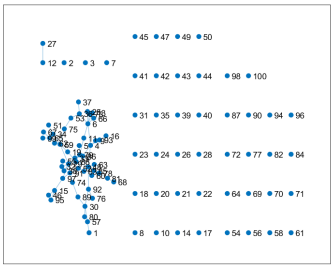


(a) Codon Usage Dataset



(b) Gas Dataset

(c) Facebook Dataset



(d) Activity Dataset

Figure 9: Connectivity Patterns from NM Regularization. Upper triangular adjacency matrices (simplified for visualization) show sparse, biologically plausible connectivity.

J THE PROOF OF THEOREM 3.2

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

$$\frac{\partial J_{NM}}{\partial w_{ij}} = \frac{\partial J}{\partial w_{ij}} + \alpha r_{ij} w_{ij}, \tag{15}$$

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

$$w_{ij} \leftarrow w_{ij} - \eta(\frac{\partial J}{\partial w_{ij}} + \alpha r_{ij} w_{ij}), \tag{16}$$

where η is the learning rate. □

K THE PROOF OF THEOREM 3.3

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

$$w_{:i} \sim \mathcal{N}(0, \tau^2 diag(r_{:i})^{-1}), \tag{17}$$

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

$$\arg\min_{w_{:i}} \{\frac{1}{2\delta^2}\|y - Xw_{:i}\|^2 + \sum \frac{r_{ji}}{2\tau^2}\|w_{ji}\|^2\}, \tag{18}$$

which equivalent to NM regularization as

$$\arg\min_{w_{:i}} \{\|y - Xw_{:i}\|^2 + \alpha \sum r_{ji}\|w_{ji}\|^2\}. \tag{19}$$

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 3.4

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:

$$P(w) = 2\Phi(\frac{w}{\Delta}) - 1. \tag{20}$$

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:

$$\|w_t - \tilde{w}_t\|^2 < \|w_0 - \tilde{w}_0\|^2. \tag{21}$$

where w_0 is the initial weight.

Follow (Tao Lin, 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.