BEYOND DIRECTED ACYCLIC COMPUTATION GRAPH WITH CYCLIC NEURAL NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

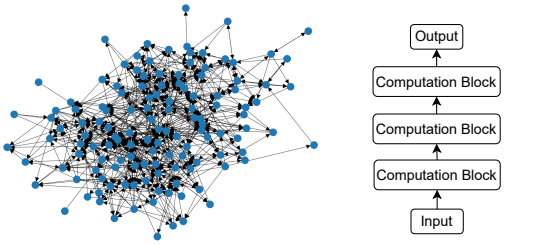
This paper investigates a fundamental yet overlooked design principle of artificial neural networks (ANN): We do not need to build ANNs layer-by-layer sequentially to guarantee the Directed Acyclic Graph (DAG) property. Inspired by biological intelligence, where neurons form a complex, graph-structured network, we introduce the transformative Cyclic Neural Networks (Cyclic NN). It emulates biological neural systems' flexible and dynamic graph nature, allowing neuron connections in any graph-like structure, including cycles. This offers greater flexibility compared to the DAG structure of current ANNs. We further develop the Graph Over Multi-layer Perceptron, the first detailed model based on this new design paradigm. We experimentally validate the advantages of Cyclic NN on widely tested datasets in most generalized cases, demonstrating its superiority over current layer-by-layer DAG neural networks. With the support of Cyclic NN, the Forward-Forward training algorithm also firstly outperforms the current Back-Propagation algorithm. This research illustrates a transformative ANN design paradigm, a significant departure from current ANN designs, potentially leading to more biologically similar ANNs.

1 INTRODUCTION

Artificial intelligence (AI) has reshaped our daily lives and is expected to have a much greater impact in the foreseeable future. Lying behind the most profound AI applications (Silver et al., 2017; OpenAI, 2023; Ramesh et al., 2021; Jumper et al., 2021), artificial neural networks (ANN) such as multi-layer perception (MLP) (Rumelhart et al., 1986), convolution neural network (CNN) (LeCun et al., 1995) and Transformer (Vaswani et al., 2017) are designed specifically for different domains to fit the training data. Regardless of the structure, they are stacked layer-by-layer to form deep ANNs for greater learning capacity. It has been a *de facto* practice until now that data is first fed into the input layer and then propagated through all the stacked layers to obtain the final representations at the output layer. This paper seeks to answer a fundamental question in ANNs: "Do we really need to stack neural networks layer-by-layer sequentially?"

To answer this question, let's first examine the evidence from biological intelligence (BI). Neuroscientists have studied the biological neurons for decades. The connectome of C. elegans is the most thoroughly studied biological neural system, and biologists depicted the most detailed connection between 302 biological neurons (White et al., 1986; Cook et al., 2019) as shown in Figure 1(a). Rather than being stacked layer-by-layer, all the neurons form a complicated connection graph, where each can connect to several other neurons within the system. We cannot even determine which neuron serves as the input/output within the neural system to process information. The same findings have also been observed in the latter more complicated neural systems, such as the biology neural connectome of drosophila larva (Winding et al., 2023), zebrafish (Brooks et al., 2022), mouse (Sporns & Bullmore, 2014) and the human brain (Shapson-Coe et al., 2024). Observed biological intelligence exhibits graph-structured, flexible, and dynamic neural systems, which are apparently different from the current layer-by-layer ANNs we build nowadays, as depicted in Figure 1(b).

The learning rules actually cause the difference in the neural system structure between BI and AI. The Hebb's Rule (Hebb, 2005), depicted as "Neurons that fire together wire together", is recognized as the fundamental learning way of biological neurons. The Spike-Timing-Dependent Plasticity (STDP) learning is then proposed to further consider the relative spiking time of pre-synapse and post-synapse neurons. Both learning rules of BI are localized, *i.e.*, the learning occurs on each neuron within its local influence scope. The localized learning rules grant the flexibility of each neuron on its connections to other neurons, which leads to the complicated graph-structured BI system. Conversely, for AI systems, the backward propagation (BP) algorithm (Rumelhart et al., 1986) has dominated the training of ANNs. Data is fed into the ANNs from the input layer, forward propagates layer by layer to the last layer, calculates a global loss for the whole ANN based on the ground-truth labels, and then reversely backward propagates the error signals layer by layer to the input layer. In this procedure, ANNs are trained by a global loss function, and the ANNs must guarantee the error from global loss can be back-propagated layer by layer. This requirement prevents current ANNs from forming cycles to ease gradient back-propagation. Current ANNs are nearly all DAG structured. To mitigate the biological implausible nature of the BP algorithm, the forward-forward (FF) algorithm (Hinton, 2022) is recently proposed to train ANNs. FF algorithm constructs good/bad samples and computes a loss function on each layer to differentiate between these samples. Similar to Hebb's Rule and STDP learning, the FF algorithm is a localized learning method. These advancements have allowed the training of ANNs to no longer rely solely on layer-by-layer back-propagation to design non-DAG-structured Cyclic Neural Networks.



(a) Connectome of BI (C. elegans) (b) Computation structure of AI

Figure 1: Neuron connection between Biology Neural Network and Artificial Neural Network

Cyclic NN distinguishes itself from the current layer-by-layer ANNs in several aspects. 1) More flexible neuron connections. Cyclic NN greatly increases the design space of ANNs beyond the DAG structure. In Cyclic NN, the information flow is not as unidirectional as in DAG. Former neurons can also adjust based on the information encoded by the latter neurons, which largely enhances information communication within the network. The flexible connection design also makes Cyclic NN more like the biological neural system. 2) Localized training. Instead of current dominating global loss-guided BP-based training, Cyclic NN is based on localized training, *i.e.*, each neuron is optimized with its own local loss function. There is no gradient propagating between neurons. Localized training has its unique advantages. It frees the need to build DAG dependency between neurons, which is the bedrock of supporting cycles within the network. Also, each neuron is optimized independently without waiting for gradients from the latter layers. 3) Computational neuron. Different from current ANNs that a neuron is considered as a d dimension to 1 dimension vector mapping; the neuron within Cyclic NN is considered the computational neuron with greater computation capacity because it is the optimization unit to fit the local task, which requires more parameters. This paper uses a linear layer to parameterize each computational neuron to fit the local classification task. It is also evident by the study of biological neuron (Beniaguev et al., 2021), which empirically proves the learning capacity of a biological neuron is much larger than a d dimension to 1 dimension vector mapping function as the neuron defined within current ANNs. We take this observation and propose the computational neuron in Cyclic NN with more capable computation to fit the local optimization task. In summary, our contributions can be summarized as follows:

- Conceptually, we compare BI and AI to investigate a fundamental yet overlooked design principle: We do not need to satisfy the DAG constraint when designing ANNs.
- Methodologically, we propose the transformative Cyclic NN, a novel ANN design paradigm that supports a much more flexible connection between neurons, which discards current directed acyclic computation graph constraints.

- We test the novel design paradigm on the most generalized case and propose Graph Over Multi-layer Perceptron, the first detailed model based on Cyclic NN.
- Experimentally, we demonstrate the advantage of the proposed Cyclic NN on widely tested datasets. At the same time, we are the first to beat the current dominating BP training using the FF training algorithm by the supported flexible network design proposed in this paper.

2 Cyclic Neural Network

One Cyclic NN model is one graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = (N_1, N_2, ..., N_{|\mathcal{V}|})$ is the computational neuron set and $\mathcal{E} = (S_1, S_2, ..., S_{|\mathcal{E}|})$ is the synapse set denoting the connections among neurons. Similar to the BI system, N_i ($\forall i \in \{1, 2, \cdots, |\mathcal{V}|\}$) is the neuron that tackles the detailed computation, and S_j ($\forall j \in \{1, 2, \cdots, |\mathcal{E}|\}$) is the synapse that propagates information between computational neurons. In the Cyclic NN, computational neurons can be connected flexibly in any way, like the BI system.

2.1 Computational Neuron

Computational neuron N acts as the computation/optimization unit in Cyclic NN. Different from the neuron in current ANNs, which indicates a d_{input} to 1 mapping, we grant N with stronger computation power as it is the optimization unit to fit the local task. It is motivated by the research that proves the computation power of a single biological neuron is similar to an MLP (Beniaguev et al., 2021). In Cyclic NN, N is parameterized by a function $f_N^{\mathbb{R}^{d_{\text{in}}^N} \to \mathbb{R}^{d_{\text{out}}^N}}(\mathbf{h}_{\text{in}}^N) = \mathbf{h}_{\text{out}}^N$ that maps from a d_{in}^N-dimensional representation $\mathbf{h}_{\text{in}}^N$ to a d_{out}-dimensional representation $\mathbf{h}_{\text{out}}^N$. d_{in}^N is the input dimension that is decided by the output of its pre-synapse neurons, and d_{out}^N is the output dimension of N. Similar to the biological neurons, each computational neuron functions as a computation/optimization unit. The computational neuron is also more independent during optimization.

2.2 Synapse

In neuroscience, synapses stand as pivotal junctions, orchestrating the complex symphony of neural communication. They serve as the critical interface between neurons, facilitating the transmission of information through chemical and electrical signals. In a Cyclic NN, we model the synapse S as the edge between neurons defined in Section 2.1. Each synapse $S_{1,2} = (N_1 \to N_2)$ is a directional edge from computational neuron N_1 to N_2. It indicates the output of N_1, $\mathbf{h}_{\text{out}}^{N_1}$, will be propagated to N_2 as part of its input $\mathbf{h}_{\text{in}}^{N_2}$. Different from the current ANNs with DAG structure, the synapses between neurons can be organized as any connected graph structure, including the cyclic graph.

2.3 Local Optimization

Local optimization is the bedrock to support Cyclic NN, which is also a distinguishing point compared with current ANNs. For current ANNs, inputs are propagated through computational neurons to obtain the final representation. Designing a global loss function $\mathcal{L}_{\text{global}}$ based on the final representation and ground-truth labels is the *de facto* practice to train ANNs. $\mathcal{L}_{\text{global}}$ is optimized with BP algorithm to propagate the error signal layer-by-layer, which also prohibits the formation of computation cycle. Conversely, Cyclic NN depends on local optimization, *i.e.*, each computational neuron is optimized locally without gradients propagated from other computational neurons. For the computational neuron N, Cyclic NN has a local loss function $\mathcal{L}_{\text{Local}}$ to optimize its parameters. The local optimization principle is similar to the BI system, where each neuron can learn from its local context.

2.4 Inference

During the inference phase, we design a readout layer to gather the output of all the neurons within the model as $\mathbf{h}_{\text{readout}} = f_{\text{readout}}([\mathbf{h}_{\text{out}}^{N_1}, \mathbf{h}_{\text{out}}^{N_2}, ..., \mathbf{h}_{\text{out}}^{N_{\mathcal{V}}}])$. $\mathbf{h}_{\text{readout}}$ collects all the encoded information and acts as the final

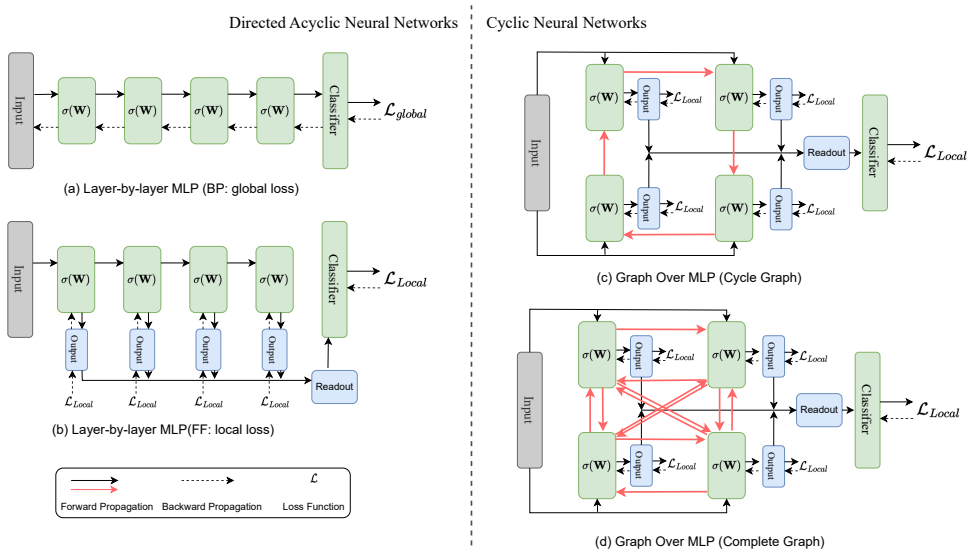


Figure 2: Comparison between different types of MLP structure.

representation for the inference task. For example, $\mathbf{h}_{\text{readout}}$ can be fed into a classifier for the classification task, where the classifier is trained together with computational neurons.

3 Graph Over Multi-Layer Perceptron

We propose the first Cyclic NN under the most generalized case, Graph Over Multi-Layer Perceptron (GOMLP), to show the design principle of Cyclic NN. As shown in Figure 2(c) and (d), GOMLP is designed by building a graph structure over the multi-layer perception to solve the classification task [1].

3.1 Input Construction

For the classification task, each sample is symbolized as the feature-label pair $(\mathbf{h}_i, y_i)$, where $\mathbf{h}_i$ is the representation of sample i and y_i is the corresponding label. To enable the local optimization illustrated in Section 3.4, a fusion function is used to construct the input as:

$$\begin{aligned} \mathbf{h}_{\text{pos}} &= f_{\text{fusion}}(\mathbf{h}, \mathbf{y}_{\text{true}}) = \mathbf{h}||\mathbf{y}_{\text{true}}, \\ \mathbf{h}_{\text{neg}} &= f_{\text{fusion}}(\mathbf{h}, \mathbf{y}_{\text{false}}) = \mathbf{h}||\mathbf{y}_{\text{false}}, \\ \mathbf{h}_{\text{neu}} &= f_{\text{fusion}}(\mathbf{h}, \mathbf{y}_{\text{neutral}}) = \mathbf{h}||\mathbf{y}_{\text{neu}}, \end{aligned} \tag{1}$$

$\mathbf{h}_{\text{pos}}$, $\mathbf{h}_{\text{neg}}$, and $\mathbf{h}_{\text{neu}}$ are the constructed input for local optimization of different parts. f_{fusion} is a function to fuse information between feature and label, which is defined as a concat function ($||$). $\mathbf{y}_{\text{true}}$ is the one-hot vector of ground-true label, $\mathbf{y}_{\text{false}}$ is the one-hot vector of a randomly sampled false label. For $\mathbf{y}_{\text{neu}}$, we place an $\frac{1}{\text{Class Number}}$ on all the dimensions of one-hot vector to indicate $\mathbf{h}_{\text{neutral}}$ is neutral to all classes. f_{fusion} can be designed as any proper function to fuse information of the input feature and the label. In our study, we design it as a simple concat function same as (Hinton, 2022).

3.2 Computation Graph

The computation graph $\mathcal{G}$ contains the computational neurons $\mathcal{V}$ and the synapses $\mathcal{E}$. Each computational neuron $N \in \mathcal{V}$ is a local module for calculation and optimization, while synapse S defines how the information

[1] Code is released at https://anonymous.4open.science/r/Cyclic-Neural-Network-025F/README.md

propagates between computational neurons. $\mathcal{G}$ can be defined as a graph generator:

$$\mathcal{G} = \text{Generator}(|\mathcal{V}|, |\mathcal{E}|). \tag{2}$$

The above-generated graph $\mathcal{G}$ denotes a general graph structure. Meanwhile, to justify the effectiveness of the proposed Cyclic NN, we test multiple graph generators in this paper.

- Chain graph. Neurons are organized layer-by-layer as shown in Figure 2(b). In this case, GOMLP degrades to Hinton's method (Hinton, 2022).
- Cycle graph. Neurons form a cycle by connecting the neurons head-to-tail as shown in Figure 2(c).
- Complete graph. Each neuron connects to all the other neurons, as shown in Figure 2(d).
- Watts-Strogatz (WS) graph (Watts & Strogatz, 1998). It produces graphs with small-world properties, including short average path lengths and high clustering.
- Barabási–Albert (BA) graph (Albert & Barabási, 2002). It generates random scale-free networks using a preferential attachment mechanism.

3.2.1 Neuron Update

In GOMLP, each neuron is parameterized by a linear layer. At each propagation, neuron N is updated by (we omit the notation of N in equation for simplicity):

$$\mathbf{h}_{\text{out}} = \sigma(\mathbf{W}\tilde{\mathbf{h}}_{\text{in}}), \tag{3}$$

where σ is the Relu activation function (Nair & Hinton, 2010), $\mathbf{W} \in \mathbb{R}_{d^N_{\text{out}} \times d^N_{\text{in}}}$ is N's parameter. d^N_{out} is N's output dimension, and d^N_{in} is N's input dimension, which is defined by the output of N's pre-synapse neurons. $\tilde{\mathbf{h}}_{\text{in}}$ is the normalized input as $\tilde{\mathbf{h}}_{\text{in}} = \frac{\mathbf{h}_{\text{in}}}{\|\mathbf{h}_{\text{in}}\|_2}$, where $\mathbf{h}_{\text{in}}$ is computational neuron N's input.

3.2.2 Synapse Propagation

Each synapse $S = (N_i \rightarrow N_j)$ is a directional edge from computational neuron N_i to N_j, which indicates N_i is the pre-synapse neuron of N_j and $\mathbf{h}^{N_i}_{out}$ (the output of N_i) will be propagated to N_j. Assume for neuron N, we obtain a set of pre-synapse neurons $(N_1, N_2, ..., N_n)$ based on the topology of $\mathcal{G}$. Then, in each propagation, N receives the output of all its pre-synapse neurons along the synapses and fuse the information to form its input by a concatenation function:

$$\mathbf{h}_{\text{in}} = \mathbf{h}||\mathbf{h}^{N_1}_{\text{out}}||\mathbf{h}^{N_2}_{\text{out}}||, ..., ||\mathbf{h}^{N_n}_{\text{out}}, \tag{4}$$

where $||$ is the concat function, $\mathbf{h}$ is the input representation constructed in Section 3.1. Then we can obtain $\mathbf{h}_{\text{in,pos}}$, $\mathbf{h}_{\text{in,neg}}$, $\mathbf{h}_{\text{in,neu}}$ by providing $\mathbf{h}_{\text{pos}}$, $\mathbf{h}_{\text{neg}}$, $\mathbf{h}_{\text{neu}}$ separately. As we relax the layer-by-layer restriction, the differentiation between the input/hidden/output layers is also relaxed. We directly put the input $\mathbf{h}$ to all computational neurons. Thus, the input dimension size of N, $d^N_{\text{in}} = d_{\mathbf{h}} + d^{N_1}_{\text{out}} + d^{N_2}_{\text{out}} + ... + d^{N_n}_{\text{out}}$.

3.3 Readout Layer

The readout layer collects information from all computational neurons. The input of the readout layer is the concat function of all computational neurons as $\mathbf{h}^{\text{readout}}_{\text{in}} = f_{\text{readout}}(\mathbf{h}^{N_*}_{\text{out}}) = ||^{|\mathcal{V}|}_{i=1}(\mathbf{h}^{N_i}_{\text{out}})$, where $||$ is the concat function. Then, the readout layer casts the representation to output dimension:

$$\hat{\mathbf{y}} = \text{Softmax}(\mathbf{W}_{\text{readout}}\mathbf{h}^{\text{readout}}_{\text{in}}), \tag{5}$$

where $\mathbf{W}_{\text{readout}} \in \mathbb{R}^{\text{Class Number} \times d(\mathbf{h}^{\text{readout}}_{\text{in}})}$ is the parameter of the readout layer and $\hat{\mathbf{y}}$ is the prediction vector.

3.4 LOCAL OPTIMIZATION

GOMLP comprises two parts that hold parameters: the computational neurons, and the readout layer. All parameters are optimized locally without gradient propagates between each parts. The computational neuron and readout layer are optimized differently with different inputs, which is shown in Algorithm 1, which contains the optimization of each computational neuron and the readout layer. The hyper-parameter T indicates the number of synapse propagation over $\mathcal{G}$.

3.4.1 COMPUTATIONAL NEURON OPTIMIZATION

Computational neurons are optimized to differentiate the positive examples from negative ones. For computational neuron N, its optimization involves $\mathbf{h}_{\text{in,pos}}$ and $\mathbf{h}_{\text{in,neg}}$. After the computational neuron update (Equation 3), we can get $\mathbf{h}_{\text{out,pos}}$ and $\mathbf{h}_{\text{out,neg}}$, respectively. Then, following (Hinton, 2022), a goodness function is used to calculate the goodness score as $p(\mathbf{h}) = \sigma(\sum_i h_i^2 - \theta * d(\mathbf{h}))$, where $p(\mathbf{h})$ is the goodness score of $\mathbf{h}$, $d(\mathbf{h})$ is the dimension size of $\mathbf{h}$, σ is the Relu activation function and θ is the threshold hyper-parameter. The binary cross-entropy loss is used to optimize each computational neuron:

$$\mathcal{L}_N = -\frac{1}{|\mathcal{D}|}\sum_{\mathcal{D}}(\log(p(\mathbf{h}_{\text{out,pos}})) - \log(p(\mathbf{h}_{\text{out,neg}}))), \tag{6}$$

where $\mathcal{D}$ is the dataset. The optimization of computational neurons aims to increase the neuron's output for positive samples while decreasing the neurons' output for negative samples. It enables each computational neuron its own ability to differentiate positive examples from negative ones.

3.4.2 READOUT LAYER OPTIMIZATION

The readout layer is designed to accomplish the classification task for GOMLP. It reads the information from all computational neurons, and makes the decision over classes. To relieve the label leakage issue, the readout layer is only optimized with $\mathbf{h}_{\text{neutral}}$. It is optimized by a multi-class cross-entropy loss:

$$\mathcal{L}_{\text{Readout}}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{|\mathcal{D}|}\sum_{|\mathcal{D}|}\sum_{c=1}^{C} y_c \log(\hat{y}_c), \tag{7}$$

where C is the number of classes, $\mathbf{y}$ is the one-hot vector of ground-truth label and $\hat{\mathbf{y}}$ is the prediction from Equation 5. Though the optimization of computational neuron and readout layer are localized and different, these two parts complement each other. Computational neurons aim to extract the hidden representations, and the readout layer aims to make the final decision.

Algorithm 1 Cyclic NN Framework Optimization

Input: dataset $\mathcal{D}$
Parameter: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathbf{W}_{\text{readout}}$
Output: $\mathcal{G}$, $\mathbf{W}_{\text{readout}}$

```
while Not Converged do
    Obtain inputs by Eq. 1 from D
    Let t = 0
    while t < T do
        for N ∈ V do
            Synapse Propagate by Eq. 4
            Computational neuron Update by Eq. 3
            Optimize N by Eq. 6
        end for
        t = t + 1
    end while
    Calculate the output of Readout layer by Eq. 5
    Optimize W_readout by Eq. 7
end while
return G = (V, E), W_readout
```

During the inference time, we pair each test sample with the neutral label to construct $\mathbf{h}_{\text{neu}}$. It then propagates through the GOMLP to obtain its representation on each computational neuron. Finally, we predict its class with the largest logit from the output of the readout layer.

3.5 COMPLEXITY ANALYSIS

Assume we have a Cyclic NN represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each $v \in \mathcal{V}$ is a linear layer and each $e \in \mathcal{E}$ represents an edge between two linear layers. To make the analysis easier, let's firstly assume the

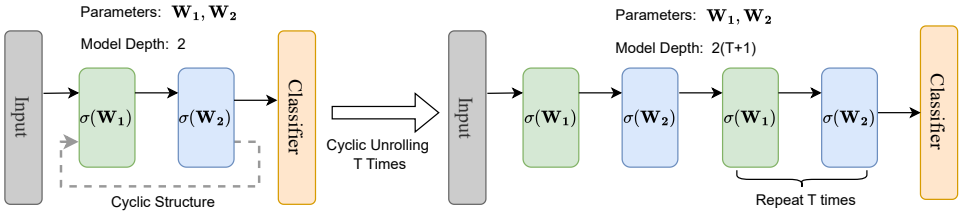


Figure 3: Cycic Structure implicitly leads to deeper model with no extra parameter cost.

time complexity for each computation neuron is $O(1)$. Then given timestep T, the input goes within the graph structure and cycles along all edges within $\mathcal{E}$ for T times, which takes $O(T * |\mathcal{E}|)$. Within each cycle, a $O(|\mathcal{V}|)$ is needed to calculate both the forward/backward pass within each computation block. The total time complexity during each step training is $O(T * |\mathcal{E}| * |\mathcal{V}|)$. Inference time complexity is the same as we only omit the backward pass. By comparison, current layer-by-layer structure is a special case for Cyclic NN when the graph is organized as chain graph and enabling back-propagation between layers to remove T. Under this case, the time complexity of current layer-by-layer form would be $O(|\mathcal{V}|)$.

To be noted that, experiment in Section 4.3 finds the timestep T is usually very small (≤ 5 in all tested datasets). So the complexity of proposed Cyclic NN can be $O(|\mathcal{E}| * |\mathcal{V}|)$ in practice. Besides, Cyclic NN is also optimized with local losses on each computation block, which frees the backward lock problem Dean et al. (2012); Löwe et al. (2019) inherited within the Backward-Propagation algorithm. Cyclic NN enables asynchronous parallel update of all computation neurons $\mathcal{V}$ at the same time. So the time complexity of Cyclic NN can be further reduced to $O(|\mathcal{E}|)$ in practice.

Let's take the example of GOMLP and further consider the time complexity of each computation neuron. The maximum complexity for each computation neuron is $O(|\mathcal{V} - 1|d * d) = O(|\mathcal{V}|d^2)$ when it receives information from all the other computation neurons. So the total time complexity of GOMLP is $O(|\mathcal{E}||\mathcal{V}|d^2)$.

3.6 Analysis on Advantage of Cyclic Structure

With piecewise linear function (such as ReLu Nair & Hinton (2010) used in this work) as the activation function after each layer, the neural network splits the whole input space as different linear regions, and the network expressiveness can be quantitatively measured by the maximal number of those regions Montufar et al. (2014); Raghu et al. (2017). More linear regions can approximate more complex functions expressed by the network. Previous research has proved a rectifier neural network with n_0 input units and L hidden layers of width $n \geq n_0$ can compute functions that have $\Omega((\frac{n}{n_0})^{(L-1)n_0} n^{n_0})$ linear regions Montufar et al. (2014). It shows neural network depth has an exponential advantage impact on its expressive power.

Now let's consider the impact of the proposed cyclic structure within network. As shown in Figure 3, there is a neural network consisting of two hidden layers parameterized by $\mathbf{W}_1$ and $\mathbf{W}_2$. The model depth without cyclic structure is 2 for two hidden layers. Simply adding a feedback edge from the second layer to the first layer, we can obtain a cyclic structure as $\sigma(\mathbf{W}_1) \leftrightarrows \sigma(\mathbf{W}_2)$. To prevent the infinite recursion, we pre-define the number of cycles as T. If we unroll the cyclic structure as shown in Figure 3, we can obtain a network as $\sigma(\mathbf{W}_1) \rightarrow \sigma(\mathbf{W}_2) \rightarrow \underbrace{\sigma(\mathbf{W}_1) \rightarrow \sigma(\mathbf{W}_2)}_{\text{Cycle T Times}}$. It is equivalent to a neural network of depth $2(T+1)$ with parameter sharing across layers. When $T = 0$, our method deteriorates to the original two-layer MLP case. When $T > 1$, the unrolled layers will split the space into more linear regions for better expressiveness. At the same time, we can see the expressiveness of neural network consists of $2(T+1)$ layers without parameter sharing as the expressiveness upper-bound of our method. In summary, the cyclic structure benefits model expressiveness by implicitly leading to deeper models without extra parameter cost for additional layers.

Table 1: Error rate (%) ↓ on different datasets.

Train	Graph	MNIST	NewsGroup	IMDB
MLP-Ensemble	-	$1.91_{\pm 0.21}$	$45.35_{\pm 0.84}$	$17.36_{\pm 0.23}$
BP	Chain*	$1.77_{\pm 0.16}$	$42.11_{\pm 0.92}$	$\mathbf{17.16}_{\pm 0.19}$
FF	Chain	$1.83_{\pm 0.2}$	$43.88_{\pm 0.28}$	$18.75_{\pm 0.92}$
BP	Chain	$1.74_{\pm 0.11}$	$38.85_{\pm 0.42}$	$17.27_{\pm 0.13}$
FF	Cycle	$1.80_{\pm 0.14}$	$43.54_{\pm 0.41}$	$18.97_{\pm 0.49}$
FF	WSGraph	$1.70_{\pm 0.17}$	$38.28_{\pm 0.13}$	$17.93_{\pm 0.28}$
FF	BAGraph	$1.64_{\pm 0.08}$	$38.41_{\pm 0.14}$	$18.20_{\pm 0.67}$
FF	Complete	$\mathbf{1.54}_{\pm 0.05}$	$\mathbf{38.266}_{\pm 0.06}$	$17.58_{\pm 0.20}$

4 EXPERIMENTS

4.1 BASELINES

We compared GOMLP with a variant of different methods to reveal the advantages of graph-structured MLP, which can be differentiated by two attributes (Training and Graph). Training indicates the training method, where BP indicates Backward Propagation (Rumelhart et al., 1986) and FF indicates the Forward-forward algorithm (Hinton, 2022). The graph indicates the graph structure of computational neurons. We keep 4 computation neurons for all methods during the experiments. The special cases are further illustrated as:

- BP-Chain*: Layer-by-layer networks trained with BP as depicted in Figure 2(a). It is the current default way of building and training ANNs.
- FF-Chain: Layer-by-layer networks trained with FF as depicted in Figure 2(b) same as (Hinton, 2022).
- BP-Chain: A modified version of BP-Chain*, where we use the structure of Figure 2(b) and trained with BP. It adds direct local supervision on each layer.

FF-Cycle, FF-WSGraph, FF-BAGraph, and FF-Complete are different versions of GOMLP, where the training is FF and only the graph generator defined in Eq. 2 differs.

4.2 OVERALL COMPARISON

The overall experiment result is shown in Table 1. We can have several interesting and exciting findings:

- FF-Complete achieves the best performance on MNIST and NewsGroup datasets and comparable results to the best one on the IMDB dataset. It is the first FF-trained model that outcompetes the BP-trained model. It is an exciting observation of the effectiveness of the FF algorithm compared with the BP algorithm.
- FF-Chain performs worse than BP-Chain* on all datasets. This observation is on par with (Hinton, 2022), where the FF lags behind the BP training algorithm when they both follow layer-by-layer organization as a chain graph. However, we can surpass BP-Chain* when organizing the computational neurons as a graph structure. This finding inevitably reveals the advantages of GOMLP by organizing multi-layer perceptron as a flexible graph structure.
- FF-Cycle achieves similar performance with FF-Chain on three datasets. It is reasonable because these two methods have only one edge difference. When we build more complex graphs (WSGraph, BAGraph, Complete Graph), we can observe much better performance immediately. It shows the benefits of enriching the communication between computational neurons by the GOMLP.

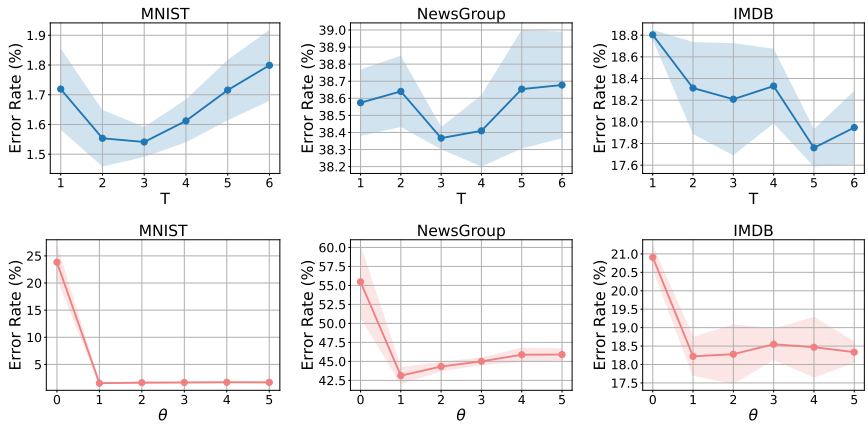


Figure 4: Parameter sensitivity of T and θ

- BP-Chain is better than BP-Chain* in most cases. Compared with BP-Chain*, BP-Chain further adds layer-wise optimization directly from the final loss. It indicates the advantageous layer-wise optimization, which provides new guidelines when designing layer-by-layer neural networks.

In summary, the experiment results answer that we do not need to stack neural networks layer-by-layer sequentially, and we can organize the neural networks as a flexible, complex graph structure like the brain. More excitingly, we can outperform the current *de facto* layer-by-layer neural network design paradigm with the Cyclic NN, and provide a totally new way of building ANNs.

4.3 Hyper-parameter Sensitivity

This section tests the impact of introduced hyper-parameters (T and θ). Results are shown in Figure 4. T controls the number of propagation between computational neurons. Larger T indicates more times the information is propagated. We can observe an error rate trend that first decreases and then increases on all three datasets. It indicates that the computational neurons need a suitable propagation number. When T is small, computational neurons can not draw sufficient lessons from each other. When T is large, computational neurons are over-propagated, which leads to the over-smoothing problem. Different from graph neural network (Chen et al., 2020) where the graph is data and the over-smoothing occurs among node representation, in GOMLP, the graph is the model, and the over-smoothing occurs among computational neurons.

θ controls the goodness threshold of each computational neuron. We can observe a sharp error rate decrease when θ increases from 0 to 1, and then it gets stable with larger θ. It indicates the existence of the goodness threshold matters more than the threshold value. When $\theta = 0$, there is little room to optimize the computational neuron, which can lead to the training collapse as the computational neuron can not differentiate the negative samples. When θ is larger, there is more room to optimize the goodness score toward the negative samples, as all the goodness scores under the threshold can represent a negative sample.

4.4 Ablation Study

Table 2: Error rate (%) ↓ of Ablation study.

Model	MNIST	NewsGroup	IMDB
FF-Complete	**1.54**	**38.26**	**18.20**
-$\mathcal{L}_N$	2.24	47.61	22.94
-$\mathcal{L}_{\text{Readout}}$	95.58	95.55	44.26

This section studies the impact of different optimization modules within GOMLP, including the computational neuron optimization $\mathcal{L}_N$ and readout layer optimization $\mathcal{L}_{\text{Readout}}$. We conduct experiments on the FF-Complete structure, and the results are summarized in Table 2. We can have the following observations: 1) The error rate increases when removing any

optimization module, indicating the usefulness of each component. 2) GOMLP falls to a very large error rate (nearly random guess) when removing $\mathcal{L}_{\text{Readout}}$. It is reasonable as we depend on the readout layer to complete the final classification task. Without optimization on the readout layer, GOMLP falls into random guess even with optimized computational neuron's input. 3) The error rate increases by removing $\mathcal{L}_N$. It shows the computational neuron's optimization can provide a more informative goodness score for the readout layer to complete the classification task. $\mathcal{L}_N$ and $\mathcal{L}_{\text{Readout}}$ complement each other within GOMLP.

5 Related Work

5.1 Localized Learning Algorithm

Although the end-to-end BP algorithm is the dominant training algorithm for deep neural networks, studies have revealed notable limitations in such end-to-end training with global objectives (Bengio et al., 2015; Crick, 1989). Numerous works have proposed alternative training methods to make ANNs more biologically plausible. Inspired by Hebbian theory (Hebb, 2005), Hebbian Learning (Gerstner & Kistler, 2002) updates weights locally between two active, connected neurons, ensuring long-term stability so previously learned information is not lost. Addressing the weight transportation problem, Feedback alignment methods (Lillicrap et al., 2016; Nøkland, 2016) replace downstream synaptic weights with random weights, eliminating the need for feedback weights in neurons. Unlike the two phases of the BP algorithm, Equilibrium Propagation (Scellier & Bengio, 2017) performs both inference and weight updates using only one type of computation. The approach in (Nøkland & Eidnes, 2019) reduces memory consumption and increases training parallelism by adopting subnetworks and layer-wise training. (Hinton, 2022) introduces a simple yet efficient local objective function that measures the goodness of positive and negative data to optimize ANNs locally.

The localized learning algorithm is the bedrock that supports the cyclic structure within the neural networks. In this paper firstly beats global BP training with pure localized learning algorithm based on cyclic structure.

5.2 Graph Generator

With many computational neurons as nodes, there are plenty of algorithms to generate computation graphs. The simplest model to generate a random graph is the Erdős–Rényi model (ERDdS & R&wi, 1959), which iterates all possible edges and adds them with a probability of $p \in [0, 1]$. (Batagelj & Brandes, 2005) propose a more efficient algorithm to create a graph with a linear computation cost in terms of running time and space requirement. Random Geometric graphs (Penrose, 2003) place nodes on geometric planes and form edges within a fixed distance, resembling real-world social networks. The Spectrum Graph Forge (Baldesi et al., 2018) creates a random graph that has the same eigenstructure as a given adjacency matrix. The Waxman model (Waxman, 1988) generates large-scale graphs that have properties similar to those of communication networks. K-regular graphs (STEGER & WORMALD, 1999) are simple graphs in which every node has the same degree. The Watts-Strogatz model (Watts & Strogatz, 1998) produces small-world graphs that guarantee high clustering coefficients and low averages of shortest path distances. The Barabási-Albert model (Albert & Barabási, 2002) generates graphs by continuously adding nodes that attach to existing nodes with high degrees, known as scale-free graphs. Due to the constraint of the BP training algorithm, current ANNs only fall into a very specialized case considering the graph structure (Directed Acyclic Graph). In this paper, we propose a transformative cyclic NN design paradigm to support all kinds of graph structures that are more flexible and have higher biology similarity.

6 Conclusion

This research introduces Cyclic NN, a novel ANN architecture inspired by the complex, graph-like neural networks in biological intelligence. This transformative design diverges from traditional directed acyclic ANN structures. Our findings, demonstrated through the Graph Over Multi-layer Perceptron model and validated on various datasets, showed enhanced performance over conventional DAG networks. This significant development paves the way for more flexible and biologically realistic AI systems, representing a major shift in ANN design.

REFERENCES

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

Luca Baldesi, Carter T. Butts, and Athina Markopoulou. Spectral graph forge: Graph generation targeting modularity. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 1727–1735. IEEE Press, 2018. doi: 10.1109/INFOCOM.2018.8485916. URL https://doi.org/10.1109/INFOCOM.2018.8485916.

Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 71 3 Pt 2A:036113, 2005. URL https://api.semanticscholar.org/CorpusID:509516.

Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

David Beniaguev, Idan Segev, and Michael London. Single cortical neurons as deep artificial neural networks. *Neuron*, 109(17):2727–2739, 2021.

Paul Brooks, Andrew Champion, and Marta Costa. Mapping of the zebrafish brain takes shape. *Nature Methods*, 19(11):1345–1346, 2022.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3438–3445, 2020.

Steven J Cook, Travis A Jarrell, Christopher A Brittin, Yi Wang, Adam E Bloniarz, Maksim A Yakovlev, Ken CQ Nguyen, Leo T-H Tang, Emily A Bayer, Janet S Duerr, et al. Whole-animal connectomes of both caenorhabditis elegans sexes. *Nature*, 571(7763):63–71, 2019.

Francis Crick. The recent excitement about neural networks. *Nature*, 337:129–132, 1989. URL https://api.semanticscholar.org/CorpusID:5892527.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

P ERDdS and A R&wi. On random graphs i. *Publ. math. debrecen*, 6(290-297):18, 1959.

Wulfram Gerstner and Werner M. Kistler. Mathematical formulations of hebbian learning. *Biological Cybernetics*, 87:404–415, 2002. URL https://api.semanticscholar.org/CorpusID:7703741.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

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

Ken Lang. Newsweeder: Learning to filter netnews. In *Machine learning proceedings 1995*, pp. 331–339. Elsevier, 1995.

Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):13276, 2016.

Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. *Advances in neural information processing systems*, 32, 2019.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4839–4850. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/nokland19a.html`.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL `https://doi.org/10.48550/arXiv.2303.08774`.

Mathew Penrose. *Random geometric graphs*, volume 5. OUP Oxford, 2003.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pp. 2847–2854. PMLR, 2017.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pp. 8821–8831. PMLR, 2021.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Alexander Shapson-Coe, Michał Januszewski, Daniel R Berger, Art Pope, Yuelong Wu, Tim Blakely, Richard L Schalek, Peter H Li, Shuohong Wang, Jeremy Maitin-Shepard, et al. A petavoxel fragment of human cerebral cortex reconstructed at nanoscale resolution. *Science*, 384(6696):eadk4858, 2024.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Olaf Sporns and Edward T Bullmore. From connections to function: the mouse brain connectome atlas. *Cell*, 157(4):773–775, 2014.

A. STEGER and N. C. WORMALD. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, 1999. doi: 10.1017/S0963548399003867.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

B.M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6 (9):1617–1622, 1988. doi: 10.1109/49.12889.

John G White, Eileen Southgate, J Nichol Thomson, Sydney Brenner, et al. The structure of the nervous system of the nematode caenorhabditis elegans. *Philos Trans R Soc Lond B Biol Sci*, 314(1165):1–340, 1986.

Michael Winding, Benjamin D Pedigo, Christopher L Barnes, Heather G Patsolic, Youngser Park, Tom Kazimiers, Akira Fushiki, Ingrid V Andrade, Avinash Khandelwal, Javier Valdes-Aleman, et al. The connectome of an insect brain. *Science*, 379(6636):eadd9330, 2023.

A APPENDIX

A.1 DATASET

Table 3: Dataset Statistics

Dataset	MNIST	NewsGroup	IMDB
# Training Samples	50,000	9,314	20,000
# Validation Samples	10,000	2,000	5,000
# Test Samples	10,000	7,532	25,000
# Dimensions	784	788	770
# Classes	10	20	2

We conduct experiments on three widely studied datasets from computer vision and natural language processing domains. Data statistics are shown in Table 3. For each dataset, the training and test split follows the original setting. We further extract 20% samples from the training data as validation sets to tune hyper-parameters.

- MNIST [2] (LeCun et al., 1989). It contains handwritten digits from 0-9, which is the most accessible and used datasets in the field of machine learning.
- NewsGroup [3] (Lang, 1995). It is a collection of approximately 20,000 newsgroup documents, partitioned across 20 different newsgroups. This dataset is widely used for experiments in text applications of machine learning techniques, such as text classification and text clustering.
- IMDB [4] (Maas et al., 2011). It is a movie review dataset crawled from IMDB. It is the most widely studied dataset for binary sentiment classification.

For MNIST, we directly use its flattened pixel values as the input of all methods and replace the first 10 pixels with labels as the fusion function, which is the same as (Hinton, 2022) and leads to an input dimension of $28 * 28 = 784$. For NLP datasets (NewsGroup, IMDB), we use BERT (Devlin et al., 2018) to encode the sentences into a fixed-length tensor (768) as the input. The fusion function is the concat function, which leads to an input dimension of $768 + 20 = 788$ for NewsGroup and $768 + 2 = 770$ for IMDB dataset.

A.2 EXPERIMENTAL SETTING.

We use Adam (Kingma & Ba, 2014) optimizer to train the model until it converges. Learning rate and weight decay are tuned within (0.1,0.01,0.001) and (0.0, 1e-2, 1e-4, 1e-6, 1e-8), respectively. The early stop technique is applied to avoid overfitting, where we stop training if there is no improvement on the validation set for continuous 10 epochs. We report the mean and variance on 20 experiments with different random seeds. All experiments are conducted on GeForce 4090 GPU.

A.3 ADVANTAGES AND LIMITATIONS

Optimized with local losses, Cyclic NN is a novel ANN design paradigm that goes beyond DAG constraint. It has several advantages. (1) Flexibility. Cyclic NN can build and optimize neural networks in any graph structure. It provides more flexibility when designing ANNs. (2) Denser information flow. Cyclic NN supports

[2] http://yann.lecun.com/exdb/mnist/

[3] http://qwone.com/ jason/20Newsgroups/

[4] https://ai.stanford.edu/ amaas/data/sentiment/

direct denser information flow during the forward pass with the synapse construction among computation neurons. (3) Biology similarity. As discussed in Section 1, Cyclic NN is more biology-similar to the complex graph-structured networks, paving the way to minimize our understanding gap between artificial neural networks and biology neural networks.

Cyclic NN has several limitations. Its flexibility in building graph-structured networks leads to more network complexity. Cycles within the network also pose more challenges to the interpretability. Currently, there are no suitable training frameworks specifically designed for localized optimization, which hinders the optimization speed of Cyclic NN. This limitation can be solved with the development of localized training frameworks.

A.4 Neural Network Comparison between RNN, GNN and Cyclic NN

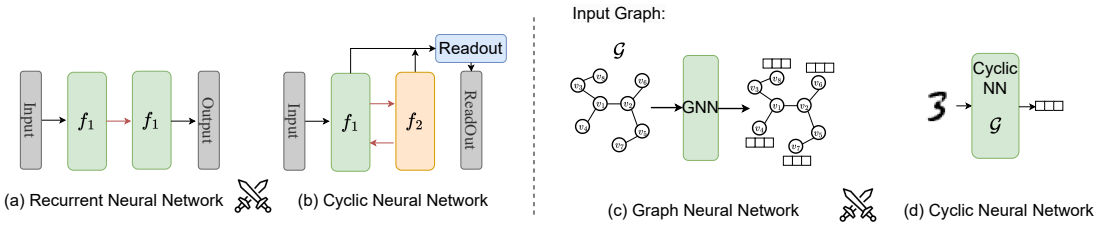


Figure 5: Neural Network Comparison between RNN, GNN and our proposed Cyclic NN.

This section makes a comparison among RNN, GNN and our newly proposed Cyclic NN. Figure 5(a) and (b) compares RNN and Cyclic NN. Recurrent structures (such as RNN, LSTM, GRU) focus on the recurrence of the same computation block. Our newly proposed Cyclic NN focus on the cyclic communication between different computation blocks as highlighted in red. The existence of cyclic structure enables building neural networks in any graph structure beyond DAG. Actually, recurrent structures can be seen as the self-loop over one computation neuron, and Cyclic NN enables much more flexible network structures beyond self-loop.

Figure 5(c) and (d) compare GNN and Cyclic NN. In GNNs (such as GCN, recurrent GNN, GAT), graph $\mathcal{G}$ is the input of network and GNN aims to learn node representation for each node. We usually use DAG structured computation within the model such as the linear layer in GCN. GNNs are the encoder of nodes within graphs, and the graph structure acts as the model's input. However, in Cyclic NN, input is not constrained to graphs. As shown in Figure 5(d), the input is one image and the Cyclic NN encodes inputs into representation. The graph structure $\mathcal{G}$ refers to the encoder itself within Cyclic NN.

A.5 Training Curve of Cyclic NN under Different Graph Structures

The training curve of Cyclic NN on different graph structures are shown in Figure 6. In Cyclic NN, the optimization occurs locally at each computation neuron and the final classifier. FF Loss is the average forward-forward loss over all computation neurons. FF loss, classifier loss and error rate changes with training epochs are shown in the figure. We can observe that for all graph structures and datasets, the decrease of losses and error rate is very stable and steady. Localized optimization focuses on optimizing parameters at a local level without propagating updates across layers. This approach helps mitigate the gradient vanishing or exploding issues commonly encountered in global optimization.

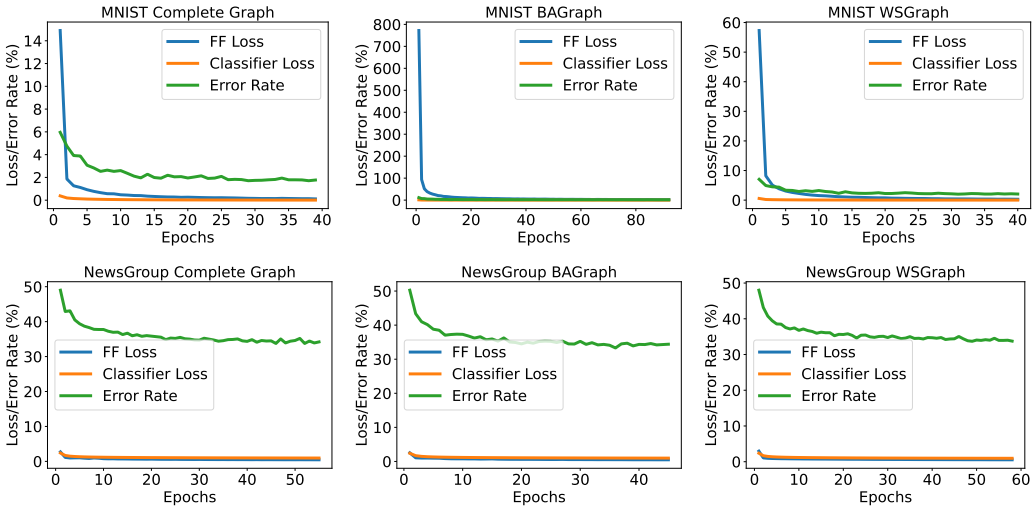


Figure 6: Training Curve of Cyclic NN under Different Graph Structure