Graph Structure of Neural Networks

Jiaxuan You [1] **Jure Leskovec** [1] **Kaiming He** [2] **Saining Xie** [2]

Abstract

Neural networks are often represented as graphs of connections between neurons. However, despite their wide use, there is currently little understanding of the relationship between the graph structure of the neural network and its predictive performance. Here we systematically investigate how does the graph structure of neural networks affect their predictive performance. To this end, we develop a novel graph-based representation of neural networks called *relational graph*, where layers of neural network computation correspond to rounds of message exchange along the graph structure. Using this representation we show that: (1) a "sweet spot" of relational graphs leads to neural networks with significantly improved predictive performance; (2) neural network's performance is approximately a smooth function of the clustering coefficient and average path length of its relational graph; (3) our findings are consistent across many different tasks and datasets; (4) the sweet spot can be identified efficiently; (5) top-performing neural networks have graph structure surprisingly similar to those of real biological neural networks. Our work opens new directions for the design of neural architectures and the understanding on neural networks in general.

1. Introduction

Deep neural networks consist of neurons organized into layers and connections between them. Architecture of a neural network can be captured by its "computational graph" where neurons are represented as nodes and directed edges link neurons in different layers. Such graphical representation demonstrates how the network passes and transforms the information from its input neurons, through hidden layers all the way to the output neurons (McClelland et al., 1986).

While it has been widely observed that performance of neural networks depends on their architecture (LeCun et al., 1998; Krizhevsky et al., 2012; Simonyan & Zisserman, 2015; Szegedy et al., 2015; He et al., 2016), there is currently little systematic understanding on the relation between a neural network's accuracy and its underlying graph structure. This is especially important for the neural architecture search, which today exhaustively searches over all possible connectivity patterns (Ying et al., 2019). From this perspective, several open questions arise: Is there a systematic link between the network structure and its predictive performance? What are structural signatures of well-performing neural networks? How do such structural signatures generalize across tasks and datasets? Is there an efficient way to check whether a given neural network is promising or not?

Establishing such a relation is both scientifically and practically important because it would have direct consequences on designing more efficient and more accurate architectures. It would also inform the design of new hardware architectures that execute neural networks. Understanding the graph structures that underlie neural networks would also advance the science of deep learning.

However, establishing the relation between network architecture and its accuracy is nontrivial, because it is unclear how to map a neural network to a graph (and vice versa). The natural choice would be to use computational graph representation but it has many limitations: (1) *lack of generality:* Computational graphs are constrained by the allowed graph properties, *e.g.*, these graphs have to be directed and acyclic (DAGs), bipartite at the layer level, and single-in-single-out at the network level (Xie et al., 2019). This limits the use of the rich tools developed for general graphs. (2) *Disconnection with biology/neuroscience:* Biological neural networks have a much richer and less templatized structure (Fornito et al., 2013). There are information exchanges, rather than just single-directional flows, in the brain networks (Stringer et al., 2018). Such biological or neurological models cannot be simply represented by directed acyclic graphs.

Here we systematically study the relationship between the graph structure of a neural network and its predictive performance. We develop a new way of representing a neural network as a graph, which we call *relational graph*. Our

[1]Department of Computer Science, Stanford University [2]Facebook AI Research. Correspondence to: Jiaxuan You <jiaxuan@cs.stanford.edu>, Saining Xie <s9xie@fb.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

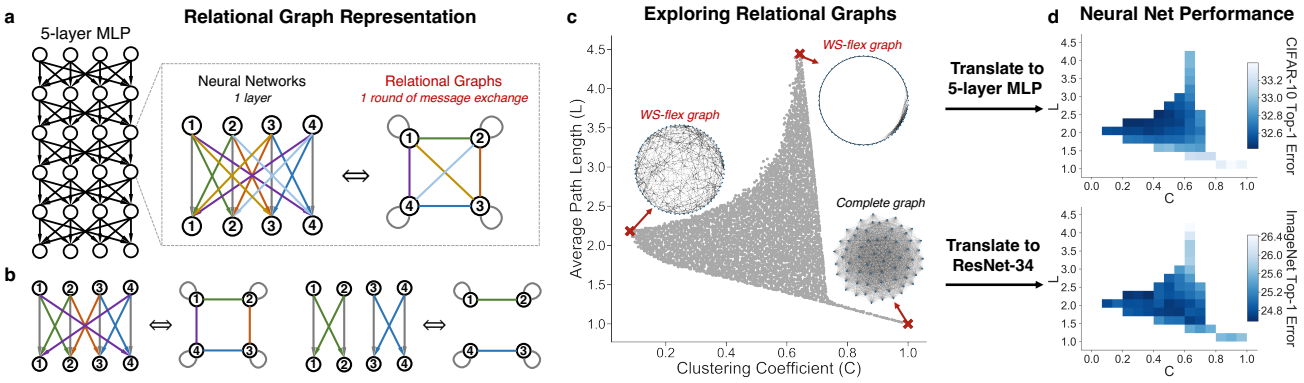

Figure 1: **Overview of our approach. (a)** A layer of a neural network can be viewed as a *relational graph* where we connect nodes that exchange messages. **(b)** More examples of neural network layers and relational graphs. **(c)** We explore the design space of relational graphs according to their graph measures, including average path length and clustering coefficient, where the complete graph corresponds to a fully-connected layer. **(d)** We translate these relational graphs to neural networks and study how their predictive performance depends on the graph measures of their corresponding relational graphs.

key insight is to focus on *message exchange*, rather than just on directed data flow. As a simple example, for a fixed-width fully-connected layer, we can represent one input channel *and* one output channel together as a single node, and an edge in the relational graph represents the message exchange between the two nodes (Figure 1(a)). Under this formulation, using appropriate message exchange definition, we show that the relational graph can represent many types of neural network layers (a fully-connected layer, a convolutional layer, *etc.*), while getting rid of many constraints of computational graphs (such as directed, acyclic, bipartite, single-in-single-out). One neural network layer corresponds to one round of message exchange over a relational graph, and to obtain deep networks, we perform message exchange over the same graph for *several rounds*. Our new representation enables us to build neural networks that are richer and more diverse and analyze them using well-established tools of network science (Barabási & Pósfai, 2016).

We then design a graph generator named WS-flex that allows us to systematically explore the design space of neural networks (*i.e.*, relation graphs). Based on the insights from neuroscience, we characterize neural networks by the clustering coefficient and average path length of their relational graphs (Figure 1(c)). Furthermore, our framework is flexible and general, as we can translate relational graphs into diverse neural architectures, including Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), ResNets, etc. with controlled computational budgets (Figure 1(d)).

Using standard image classification datasets CIFAR-10 and ImageNet, we conduct a systematic study on how the architecture of neural networks affects their predictive performance. We make several important empirical observations:

- A "sweet spot" of relational graphs lead to neural networks with significantly improved performance under controlled computational budgets (Section 5.1).
- Neural network's performance is approximately a smooth function of the clustering coefficient and average path length of its relational graph. (Section 5.2).
- Our findings are consistent across many architectures (MLPs, CNNs, ResNets, EfficientNet) and tasks (CIFAR-10, ImageNet) (Section 5.3).
- The sweet spot can be identified efficiently. Identifying a sweet spot only requires a few samples of relational graphs and a few epochs of training (Section 5.4).
- Well-performing neural networks have graph structure surprisingly similar to those of real biological neural networks (Section 5.5).

Our results have implications for designing neural network architectures, advancing the science of deep learning and improving our understanding of neural networks in general.

2. Neural Networks as Relational Graphs

To explore the graph structure of neural networks, we first introduce the concept of our *relational graph* representation and its instantiations. We demonstrate how our representation can capture diverse neural network architectures under a unified framework. Using the language of graph in the context of deep learning helps bring the two worlds together and establish a foundation for our study.

2.1. Message Exchange over Graphs

We start by revisiting the definition of a neural network from the graph perspective. We define a graph $G = (\mathcal{V}, \mathcal{E})$ by its node set $\mathcal{V} = \{v_1, ..., v_n\}$ and edge set $\mathcal{E} \subseteq \{(v_i, v_j)|v_i, v_j \in \mathcal{V}\}$. We assume each node v has a node feature scalar/vector x_v.

	Fixed-width MLP	Variable-width MLP	ResNet-34	ResNet-34-sep	ResNet-50
Node feature $\mathbf{x}_i$	Scalar: 1 dimension of data	Vector: multiple dimensions of data	Tensor: multiple channels of data	Tensor: multiple channels of data	Tensor: multiple channels of data
Message function $f_i(\cdot)$	Scalar multiplication	(Non-square) matrix multiplication	3×3 Conv	3×3 depth-wise and 1×1 Conv	3×3 and 1×1 Conv
Aggregation function $\text{AGG}(\cdot)$	$\sigma(\sum(\cdot))$	$\sigma(\sum(\cdot))$	$\sigma(\sum(\cdot))$	$\sigma(\sum(\cdot))$	$\sigma(\sum(\cdot))$
Number of rounds R	1 round per layer	1 round per layer	34 rounds with residual connections	34 rounds with residual connections	50 rounds with residual connections

Table 1: **Diverse neural architectures expressed in the language of relational graphs.** These architectures are usually implemented as complete relational graphs, while we systematically explore more graph structures for these architectures.

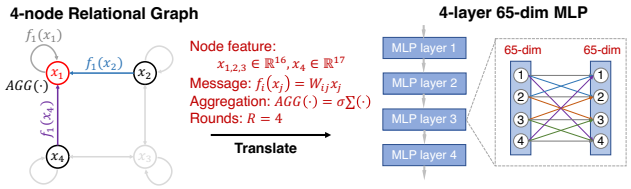

Figure 2: **Example of translating a 4-node relational graph to a 4-layer 65-dim MLP.** We highlight the message exchange for node x_1. Using different definitions of x_i, $f_i(\cdot)$, $\text{AGG}(\cdot)$ and R (those defined in Table 1), relational graphs can be translated to diverse neural architectures.

We call graph G a *relational graph*, when it is associated with message exchanges between neurons. Specifically, a message exchange is defined by a message function, whose input is a node's feature and output is a message, and an aggregation function, whose input is a set of messages and output is the updated node feature. At each round of message exchange, each node sends messages to its neighbors, and aggregates incoming messages from its neighbors. Each message is transformed at each edge through a message function $f(\cdot)$, then they are aggregated at each node via an aggregation function $\text{AGG}(\cdot)$. Suppose we conduct R rounds of message exchange, then the r-th round of message exchange for a node v can be described as

$$\mathbf{x}_v^{(r+1)} = \text{AGG}^{(r)}(\{f_v^{(r)}(\mathbf{x}_u^{(r)}), \forall u \in N(v)\}) \quad (1)$$

where u, v are nodes in G, $N(v) = \{u | v \vee (u, v) \in \mathcal{E}\}$ is the neighborhood of node v where we assume all nodes have self-edges, $\mathbf{x}_v^{(r)}$ is the input node feature and $\mathbf{x}_v^{(r+1)}$ is the output node feature for node v. Note that this message exchange can be defined over any graph G; for simplicity, we only consider undirected graphs in this paper.

Equation 1 provides a general definition for message exchange. In the remainder of this section, we discuss how this general message exchange definition can be instantiated as different neural architectures. We summarize the different instantiations in Table 1, and provide a concrete example of instantiating a 4-layer 65-dim MLP in Figure 2.

2.2. Fixed-width MLPs as Relational Graphs

A Multilayer Perceptron (MLP) consists of layers of computation units (neurons), where each neuron performs a weighted sum over scalar inputs and outputs, followed by some non-linearity. Suppose the r-th layer of an MLP takes $\mathbf{x}^{(r)}$ as input and $\mathbf{x}^{(r+1)}$ as output, then a neuron computes:

$$x_i^{(r+1)} = \sigma(\sum_j w_{ij}^{(r)} x_j^{(r)}) \quad (2)$$

where $w_{ij}^{(r)}$ is the trainable weight, $x_j^{(r)}$ is the j-th dimension of the input $\mathbf{x}^{(r)}$, *i.e.*, $\mathbf{x}^{(r)} = (x_1^{(r)}, ..., x_n^{(r)})$, $x_i^{(r+1)}$ is the i-th dimension of the output $\mathbf{x}^{(r+1)}$, and σ is the non-linearity.

Let us consider the special case where the input and output of all the layers $x^{(r)}, 1 \leq r \leq R$ have the same feature dimensions. In this scenario, a fully-connected, fixed-width MLP layer can be expressed with a *complete relational graph*, where each node x_i connects to all the other nodes $\{x_1, \ldots, x_n\}$, *i.e.*, neighborhood set $N(v) = \mathcal{V}$ for each node v. Additionally, a fully-connected fixed-width MLP layer has a special message exchange definition, where the message function is $f_i(x_j) = w_{ij} x_i$, and the aggregation function is $\text{AGG}(\{x_i\}) = \sigma(\sum\{x_i\})$.

The above discussion reveals that a fixed-width MLP can be viewed as a complete relational graph with a special message exchange function. Therefore, a fixed-width MLP is a special case under a much more general model family, where *the message function, aggregation function, and most importantly, the relation graph structure can vary*.

This insight allows us to generalize fixed-width MLPs from using complete relational graph to any general relational graph G. Based on the general definition of message exchange in Equation 1, we have:

$$x_i^{(r+1)} = \sigma(\sum_{j \in N(i)} w_{ij}^{(r)} x_j^{(r)}) \quad (3)$$

where i, j are nodes in G and $N(i)$ is defined by G.

2.3. General Neural Networks as Relational Graphs

The graph viewpoint in Equation 3 lays the foundation of representing fixed-width MLPs as relational graphs. In this

section, we discuss how we can further generalize relational graphs to general neural networks.

Variable-width MLPs as relational graphs. An important design consideration for general neural networks is that layer width often varies through out the network. For example, in CNNs, a common practice is to double the layer width (number of feature channels) after spatial down-sampling. To represent neural networks with variable layer width, we generalize node features from scalar $x_i^{(r)}$ to vector $\mathbf{x}_i^{(r)}$ which consists of some dimensions of the input $\mathbf{x}^{(r)}$ for the MLP , *i.e.*, $\mathbf{x}^{(r)} = \textsc{Concat}(\mathbf{x}_1^{(r)}, ..., \mathbf{x}_n^{(r)})$, and generalize message function $f_i(\cdot)$ from scalar multiplication to matrix multiplication:

$$\mathbf{x}_i^{(r+1)} = \sigma\Big(\sum_{j \in N(i)} \mathbf{W}_{ij}^{(r)} \mathbf{x}_j^{(r)}\Big) \tag{4}$$

where $\mathbf{W}_{ij}^{(r)}$ is the weight matrix. Additionally, we allow that: (1) the same node in different layers, $\mathbf{x}_i^{(r)}$ and $\mathbf{x}_i^{(r+1)}$, can have different dimensions; (2) Within a layer, different nodes in the graph, $\mathbf{x}_i^{(r)}$ and $\mathbf{x}_j^{(r)}$, can have different dimensions. This generalized definition leads to a flexible graph representation of neural networks, as we can reuse the same relational graph across different layers with arbitrary width. Suppose we use an n-node relational graph to represent an m-dim layer, then $m \bmod n$ nodes have $\lfloor m/n \rfloor + 1$ dimensions each, remaining $n - (m \bmod n)$ nodes will have $\lfloor m/n \rfloor$ dimensions each. For example, if we use a 4-node relational graph to represent a 2-layer neural network. If the first layer has width of 5 while the second layer has width of 9, then the 4 nodes have dimensions $\{2, 1, 1, 1\}$ in the first layer and $\{3, 2, 2, 2\}$ in the second layer.

Note that under this definition, the maximum number of nodes of a relational graph is bounded by the width of the narrowest layer in the corresponding neural network (since the feature dimension for each node must be at least 1).

CNNs as relational graphs. We further make relational graphs applicable to CNNs where the input becomes an image tensor $\mathbf{X}^{(r)}$. We generalize the definition of node features from vector $\mathbf{x}_i^{(r)}$ to tensor $\mathbf{X}_i^{(r)}$ that consists of some channels of the input image $\mathbf{X}^{(r)} = \textsc{Concat}(\mathbf{X}_1^{(r)}, ..., \mathbf{X}_n^{(r)})$. We then generalize the message exchange definition with convolutional operator. Specifically,

$$\mathbf{X}_i^{(r+1)} = \sigma\Big(\sum_{j \in N(i)} \mathbf{W}_{ij}^{(r)} * \mathbf{X}_j^{(r)}\Big) \tag{5}$$

where $*$ is the convolutional operator and $\mathbf{W}_{ij}^{(r)}$ is the convolutional filter. Under this definition, the widely used dense convolutions are again represented as complete graphs.

Modern neural architectures as relational graphs. Finally, we generalize relational graphs to represent modern neural architectures with more sophisticated designs. For example, to represent a ResNet (He et al., 2016), we keep the residual connections between layers unchanged. To represent neural networks with bottleneck transform (He et al., 2016), a relational graph alternatively applies message exchange with 3×3 and 1×1 convolution; similarly, in the efficient computing setup, the widely used separable convolution (Howard et al., 2017; Chollet, 2017) can be viewed as alternatively applying message exchange with 3×3 depth-wise convolution and 1×1 convolution.

Overall, relational graphs provide a general representation for neural networks. With proper definitions of node features and message exchange, relational graphs can represent diverse neural architectures, as is summarized in Table 1.

3. Exploring Relational Graphs

In this section, we describe in detail how we design and explore the space of relational graphs defined in Section 2, in order to study the relationship between the graph structure of neural networks and their predictive performance. Three main components are needed to make progress: (1) *graph measures* that characterize graph structural properties, (2) *graph generators* that can generate diverse graphs, and (3) a way to *control the computational budget*, so that the differences in performance of different neural networks are due to their diverse relational graph structures.

3.1. Selection of Graph Measures

Given the complex nature of graph structure, *graph measures* are often used to characterize graphs. In this paper, we focus on one global graph measure, average path length, and one local graph measure, clustering coefficient. Notably, these two measures are widely used in network science (Watts & Strogatz, 1998) and neuroscience (Sporns, 2003; Bassett & Bullmore, 2006). Specifically, average path length measures the average shortest path distance between any pair of nodes; clustering coefficient measures the proportion of edges between the nodes within a given node's neighborhood, divided by the number of edges that could possibly exist between them, averaged over all the nodes. There are other graph measures that can be used for analysis, which are included in the Appendix.

3.2. Design of Graph Generators

Given selected graph measures, we aim to generate diverse graphs that can cover a large span of graph measures, using a *graph generator*. However, such a goal requires careful generator designs: classic graph generators can only generate a limited class of graphs, while recent learning-based graph generators are designed to imitate given exemplar graphs (Kipf & Welling, 2017; Li et al., 2018; You et al., 2018a;b; 2019a).

Limitations of existing graph generators. To illustrate the limitation of existing graph generators, we investigate the following classic graph generators: (1) Erdős-Rényi (ER) model that can sample graphs with given node and edge number uniformly at random (Erdős & Rényi, 1960); (2) Watts-Strogatz (WS) model that can generate graphs with small-world properties (Watts & Strogatz, 1998); (3) Barabási-Albert (BA) model that can generate scale-free graphs (Albert & Barabási, 2002); (4) Harary model that can generate graphs with maximum connectivity (Harary, 1962); (5) regular ring lattice graphs (ring graphs); (6) complete graphs. For all types of graph generators, we control the number of nodes to be 64, enumerate all possible discrete parameters and grid search over all continuous parameters of the graph generator. We generate 30 random graphs with different random seeds under each parameter setting. In total, we generate 486,000 WS graphs, 53,000 ER graphs, 8,000 BA graphs, 1,800 Harary graphs, 54 ring graphs and 1 complete graph (more details provided in the Appendix). In Figure 3, we can observe that graphs generated by those classic graph generators have a limited span in the space of average path length and clustering coefficient.

WS-flex graph generator. Here we propose the WS-flex graph generator that can generate graphs with a wide coverage of graph measures; notably, WS-flex graphs almost encompass all the graphs generated by classic random generators mentioned above, as is shown in Figure 3. WS-flex generator generalizes WS model by relaxing the constraint that all the nodes have the same degree before random rewiring. Specifically, WS-flex generator is parametrized by node n, average degree k and rewiring probability p. The number of edges is determined as $e = \lfloor n * k/2 \rfloor$. Specifically, WS-flex generator first creates a ring graph where each node connects to $\lfloor e/n \rfloor$ neighboring nodes; then the generator randomly picks $e \bmod n$ nodes and connects each node to one closest neighboring node; finally, all the edges are randomly rewired with probability p. We use WS-flex generator to smoothly sample within the space of clustering coefficient and average path length, then sub-sample 3942 graphs for our experiments, as is shown in Figure 1(c).

3.3. Controlling Computational Budget

To compare the neural networks translated by these diverse graphs, it is important to ensure that all networks have approximately the same complexity, so that the differences in performance are due to their relational graph structures. We use FLOPS (# of multiply-adds) as the metric. We first compute the FLOPS of our baseline network instantiations (*i.e.* complete relational graph), and use them as the reference complexity in each experiment. As described in Section 2.3, a relational graph structure can be instantiated as a neural network with variable width, by partitioning dimensions or channels into disjoint set of node features. Therefore, we

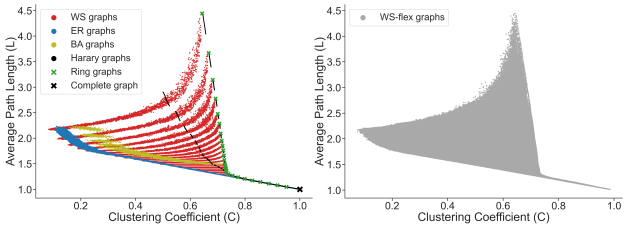

Figure 3: **Graphs generated by different graph generators.** The proposed graph generator WS-flex can cover a much larger region of graph design space. WS (Watts-Strogatz), BA (Barabási-Albert), ER (Erdős-Rényi).

can conveniently adjust the width of a neural network to match the reference complexity (within 0.5% of baseline FLOPS) without changing the relational graph structures. We provide more details in the Appendix.

4. Experimental Setup

Considering the large number of candidate graphs (3942 in total) that we want to explore, we first investigate graph structure of MLPs on the CIFAR-10 dataset (Krizhevsky, 2009) which has 50K training images and 10K validation images. We then further study the larger and more complex task of ImageNet classification (Russakovsky et al., 2015), which consists of 1K image classes, 1.28M training images and 50K validation images.

4.1. Base Architectures

For CIFAR-10 experiments, We use a 5-layer MLP with 512 hidden units as the baseline architecture. The input of the MLP is a 3072-d flattened vector of the $(32 \times 32 \times 3)$ image, the output is a 10-d prediction. Each MLP layer has a ReLU non-linearity and a BatchNorm layer (Ioffe & Szegedy, 2015). We train the model for 200 epochs with batch size 128, using cosine learning rate schedule (Loshchilov & Hutter, 2016) with an initial learning rate of 0.1 (annealed to 0, no restarting). We train all MLP models with 5 different random seeds and report the averaged results.

For ImageNet experiments, we use three ResNet-family architectures, including (1) ResNet-34, which only consists of basic blocks of 3×3 convolutions (He et al., 2016); (2) ResNet-34-sep, a variant where we replace all 3×3 dense convolutions in ResNet-34 with 3×3 separable convolutions (Chollet, 2017); (3) ResNet-50, which consists of bottleneck blocks (He et al., 2016) of 1×1, 3×3, 1×1 convolutions. Additionally, we use EfficientNet-B0 architecture (Tan & Le, 2019) that achieves good performance in the small computation regime. Finally, we use a simple 8-layer CNN with 3×3 convolutions. The model has 3 stages with [64, 128, 256] hidden units. Stride-2 convolutions are used for down-sampling. The stem and head layers are

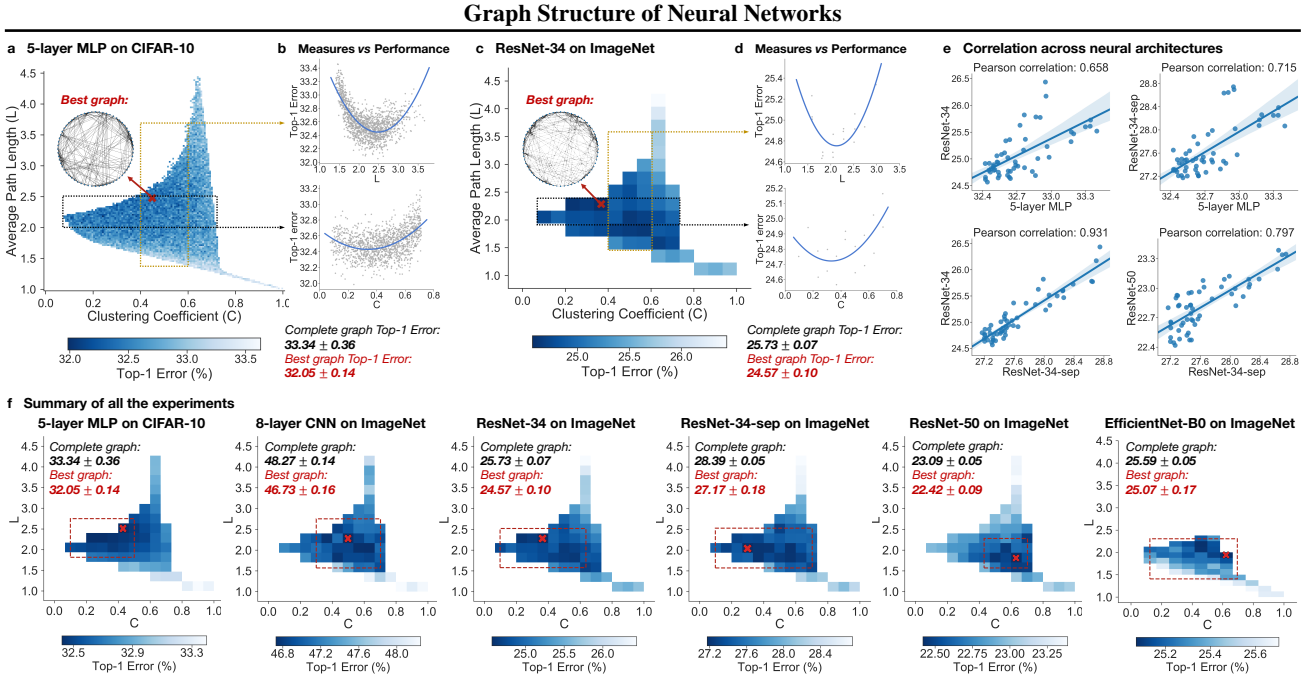

Figure 4: **Key results.** The computational budgets of all the experiments are rigorously controlled. Each visualized result is averaged over at least 3 random seeds. A complete graph with $C = 1$ and $L = 1$ (lower right corner) is regarded as the baseline. **(a)(c) Graph measures *vs.* neural network performance.** The best graphs significantly outperform the baseline complete graphs. **(b)(d) Single graph measure *vs.* neural network performance.** Relational graphs that fall within the given range are shown as grey points. The overall smooth function is indicated by the blue regression line. **(e) Consistency across architectures.** Correlations of the performance of the same set of 52 relational graphs when translated to different neural architectures are shown. **(f) Summary of all the experiments.** Best relational graphs (the red crosses) consistently outperform the baseline complete graphs across different settings. Moreover, we highlight the "sweet spots" (red rectangular regions), in which relational graphs are not statistically worse than the best relational graphs (bins with red crosses). Bin values of 5-layer MLP on CIFAR-10 are average over all the relational graphs whose C and L fall into the given bin.

the same as a ResNet. We train all the ImageNet models for 100 epochs using cosine learning rate schedule with initial learning rate of 0.1. Batch size is 256 for ResNet-family models and 512 for EfficientNet-B0. We train all ImageNet models with 3 random seeds and report the averaged performance. All the baseline architectures have a complete relational graph structure. The reference computational complexity is 2.89e6 FLOPS for MLP, 3.66e9 FLOPS for ResNet-34, 0.55e9 FLOPS for ResNet-34-sep, 4.09e9 FLOPS for ResNet-50, 0.39e9 FLOPS for EffcientNet-B0, and 0.17e9 FLOPS for 8-layer CNN. Training an MLP model roughly takes 5 minutes on a NVIDIA Tesla V100 GPU, and training a ResNet model on ImageNet roughly takes a day on 8 Tesla V100 GPUs with data parallelism. We provide more details in Appendix.

4.2. Exploration with Relational Graphs

For all the architectures, we instantiate each sampled relational graph as a neural network, using the corresponding definitions outlined in Table 1. Specifically, we replace all the dense layers (linear layers, 3×3 and 1×1 convolution layers) with their relational graph counterparts. We leave the input and output layer unchanged and keep all the other designs (such as down-sampling, skip-connections, *etc.*) intact. We then match the reference computational complexity for all the models, as discussed in Section 3.3.

For CIFAR-10 MLP experiments, we study 3942 sampled relational graphs of 64 nodes as described in Section 3.2. For ImageNet experiments, due to high computational cost, we sub-sample 52 graphs uniformly from the 3942 graphs. Since EfficientNet-B0 is a small model with a layer that has only 16 channels, we can not reuse the 64-node graphs sampled for other setups. We re-sample 48 relational graphs with 16 nodes following the same procedure in Section 3.

5. Results

In this section, we summarize the results of our experiments and discuss our key findings. We collect top-1 errors for all the sampled relational graphs on different tasks and architectures, and also record the graph measures (average path length L and clustering coefficient C) for each sampled graph. We present these results as heat maps of graph measures *vs.* predictive performance (Figure 4(a)(c)(f)).

5.1. A Sweet Spot for Top Neural Networks

Overall, the heat maps of graph measures *vs.* predictive performance (Figure 4(f)) show that there *exist* graph structures that can outperform the complete graph (the pixel on bottom right) baselines. The best performing relational graph can outperform the complete graph baseline by 1.4% top-1 error on CIFAR-10, and 0.5% to 1.2% for models on ImageNet.

Notably, we discover that top-performing graphs tend to cluster into a sweet spot in the space defined by C and L (red rectangles in Figure 4(f)). We follow these steps to identify a sweet spot: (1) we downsample and aggregate the 3942 graphs in Figure 4(a) into a coarse resolution of 52 bins, where each bin records the performance of graphs that fall into the bin; (2) we identify the bin with best average performance (red cross in Figure 4(f)); (3) we conduct one-tailed t-test over each bin against the best-performing bin, and record the bins that are *not* significantly worse than the best-performing bin (p-value 0.05 as threshold). The minimum area rectangle that covers these bins is visualized as a sweet spot. For 5-layer MLP on CIFAR-10, the sweet spot is $C \in [0.10, 0.50]$, $L \in [1.82, 2.75]$.

5.2. Neural Network Performance as a Smooth Function over Graph Measures

In Figure 4(f), we observe that neural network's predictive performance is approximately a smooth function of the clustering coefficient and average path length of its relational graph. Keeping one graph measure fixed in a small range ($C \in [0.4, 0.6]$, $L \in [2, 2.5]$), we visualize network performances against the other measure (shown in Figure 4(b)(d)). We use second degree polynomial regression to visualize the overall trend. We observe that both clustering coefficient and average path length are indicative of neural network performance, demonstrating a smooth U-shape correlation.

5.3. Consistency across Architectures

Given that relational graph defines a shared design space across various neural architectures, we observe that relational graphs with certain graph measures may consistently perform well regardless of how they are instantiated.

Qualitative consistency. We visually observe in Figure 4(f) that the sweet spots are roughly consistent across different architectures. Specifically, if we take the union of the sweet spots across architectures, we have $C \in [0.43, 0.50]$, $L \in [1.82, 2.28]$ which is the consistent sweet spot across architectures. Moreover, the U-shape trends between graph measures and corresponding neural network performance, shown in Figure 4(b)(d), are also visually consistent.

Quantitative consistency. To further quantify this consistency across tasks and architectures, we select the 52 bins in the heat map in Figure 4(f), where the bin value indicates the average performance of relational graphs whose graph measures fall into the bin range. We plot the correlation of the 52 bin values across different pairs of tasks, shown in Figure 4(e). We observe that the performance of relational graphs with certain graph measures correlates across different tasks and architectures. For example, even though a ResNet-34 has much higher complexity than a 5-layer MLP, and ImageNet is a much more challenging dataset than CIFAR-10, a fixed set relational graphs would perform similarly in both settings, indicated by a Pearson correlation of 0.658 (p-value $< 10^{-8}$).

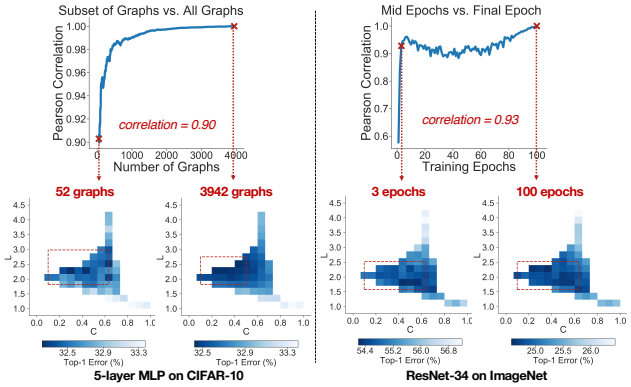

Figure 5: **Quickly identifying a sweet spot. Left:** The correlation between sweet spots identified using fewer samples of relational graphs and using all 3942 graphs. **Right:** The correlation between sweet spots identified at the intermediate training epochs and the final epoch (100 epochs).

5.4. Quickly Identifying a Sweet Spot

Training thousands of relational graphs until convergence might be computationally prohibitive. Therefore, we quantitatively show that a sweet spot can be identified with much less computational cost, *e.g.*, by sampling fewer graphs and training for fewer epochs.

How many graphs are needed? Using the 5-layer MLP on CIFAR-10 as an example, we consider the heat map over 52 bins in Figure 4(f) which is computed using 3942 graph samples. We investigate if a similar heat map can be produced with much fewer graph samples. Specifically, we sub-sample the graphs in each bin while making sure each bin has at least one graph. We then compute the correlation between the 52 bin values computed using all 3942 graphs and using sub-sampled fewer graphs, as is shown in Figure 5 (left). We can see that bin values computed using only 52 samples have a high 0.90 Pearson correlation with the bin values computed using full 3942 graph samples. This finding suggests that, in practice, much fewer graphs are needed to conduct a similar analysis.

Graph	Path (L)	Clustering (C)	CIFAR-10 Error (%)
Complete graph	1.00	1.00	33.34 ± 0.36
Cat cortex	1.81	0.55	33.01 ± 0.22
Macaque visual cortex	1.73	0.53	32.78 ± 0.21
Macaque whole cortex	**2.38**	**0.46**	32.77 ± 0.14
Consistent sweet spot across neural architectures	**1.82-2.28**	**0.43-0.50**	32.50 ± 0.33
Best 5-layer MLP	**2.48**	**0.45**	32.05 ± 0.14

Table 2: Top artificial neural networks can be similar to biological neural networks (Bassett & Bullmore, 2006).

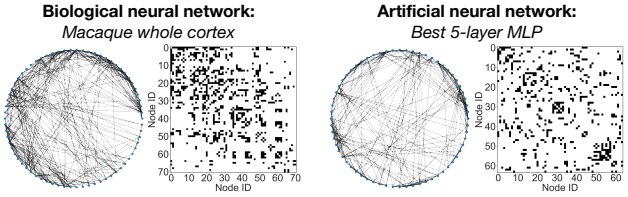

Figure 6: Visualizations of graph structure of biological (**left**) and artificial (**right**) neural networks.

How many training epochs are needed? Using ResNet-34 on ImageNet as an example, we compute the correlation between the validation top-1 error of partially trained models and the validation top-1 error of models trained for full 100 epochs, over the 52 sampled relational graphs, as is visualized in Figure 5 (right). Surprisingly, models trained after 3 epochs already have a high correlation (0.93), which means that *good relational graphs perform well even at the early training epochs*. This finding is also important as it indicates that the computation cost to determine if a relational graph is promising can be greatly reduced.

5.5. Network Science and Neuroscience Connections

Network science. The average path length that we measure characterizes how well information is exchanged across the network (Latora & Marchiori, 2001), which aligns with our definition of relational graph that consists of rounds of message exchange. Therefore, the U-shape correlation in Figure 4(b)(d) might indicate a trade-off between message exchange efficiency (Sengupta et al., 2013) and capability of learning *distributed representations* (Hinton, 1984).

Neuroscience. The best-performing relational graph that we discover surprisingly resembles biological neural networks, as is shown in Table 2 and Figure 6. The similarities are in two-fold: (1) the graph measures (L and C) of top artificial neural networks are highly similar to biological neural networks; (2) with the relational graph representation, we can translate biological neural networks to 5-layer MLPs, and found that these networks also outperform the baseline complete graphs. While our findings are preliminary, our approach opens up new possibilities for interdisciplinary research in network science, neuroscience and deep learning.

6. Related Work

Neural network connectivity. The design of neural network connectivity patterns has been focused on *computational graphs* at different granularity: the macro structures, *i.e.* connectivity across layers (LeCun et al., 1998; Krizhevsky et al., 2012; Simonyan & Zisserman, 2015; Szegedy et al., 2015; He et al., 2016; Huang et al., 2017; Tan & Le, 2019), and the micro structures, *i.e.* connectivity within a layer (LeCun et al., 1998; Xie et al., 2017; Zhang et al., 2018; Howard et al., 2017; Dao et al., 2019; Alizadeh et al., 2019). Our current exploration focuses on the latter, but the same methodology can be extended to the macro space. Deep Expander Networks (Prabhu et al., 2018) adopt expander graphs to generate bipartite structures. RandWire (Xie et al., 2019) generates macro structures using existing graph generators. However, the statistical relationships between graph structure measures and network predictive performances were not explored in those works. Another related work is Cross-channel Communication Networks (Yang et al., 2019) which aims to encourage the neuron communication through message passing, where only a complete graph structure is considered.

Neural architecture search. Efforts on learning the connectivity patterns at micro (Ahmed & Torresani, 2018; Wortsman et al., 2019; Yang et al., 2018), or macro (Zoph & Le, 2017; Zoph et al., 2018) level mostly focus on improving learning/search algorithms (Liu et al., 2018; Pham et al., 2018; Real et al., 2019; Liu et al., 2019). NAS-Bench-101 (Ying et al., 2019) defines a graph search space by enumerating DAGs with constrained sizes (≤ 7 nodes, *cf.* 64-node graphs in our work). Our work points to a new path: instead of exhaustively searching over all the possible connectivity patterns, certain graph generators and graph measures could define a smooth space where the search cost could be significantly reduced.

7. Discussions

Hierarchical graph structure of neural networks. As the first step in this direction, our work focuses on graph structures at the layer level. Neural networks are intrinsically hierarchical graphs (from connectivity of neurons to that of layers, blocks, and networks) which constitute a more complex design space than what is considered in this paper. Extensive exploration in that space will be computationally prohibitive, but we expect our methodology and findings to generalize.

Efficient implementation. Our current implementation uses standard CUDA kernels thus relies on weight masking, which leads to worse wall-clock time performance compared with baseline complete graphs. However, the practical adoption of our discoveries is not far-fetched. Complementary

to our work, there are ongoing efforts such as block-sparse kernels (Gray et al., 2017) and fast sparse ConvNets (Elsen et al., 2019) which could close the gap between theoretical FLOPS and real-world gains. Our work might also inform the design of new hardware architectures, *e.g.*, biologically-inspired ones with spike patterns (Pei et al., 2019).

Prior *vs.* Learning. We currently utilize the relational graph representation as a *structural prior*, *i.e.*, we hard-wire the graph structure on neural networks throughout training. It has been shown that deep ReLU neural networks can automatically learn sparse representations (Glorot et al., 2011). A further question arises: without imposing graph priors, does any graph structure emerge from training a (fully-connected) neural network?

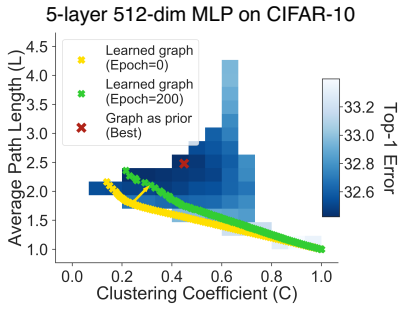

Figure 7: **Prior *vs.* Learning.** Results for 5-layer MLPs on CIFAR-10. We highlight the best-performing graph when used as a *structural prior*. Additionally, we train a fully-connected MLP, and visualize the learned weights as a relational graph (different points are graphs under different thresholds). The learned graph structure moves towards the "sweet spot" after training but does not close the gap.

As a preliminary exploration, we "reverse-engineer" a trained neural network and study the emerged relational graph structure. Specifically, we train a fully-connected 5-layer MLP on CIFAR-10 (the same setup as in previous experiments). We then try to infer the underlying relational graph structure of the network via the following steps: (1) to get nodes in a relational graph, we stack the weights from all the hidden layers and group them into 64 nodes, following the procedure described in Section 2.2; (2) to get undirected edges, the weights are summed by their transposes; (3) we compute the Frobenius norm of the weights as the edge value; (4) we get a sparse graph structure by binarizing edge values with a certain threshold.

We show the extracted graphs under different thresholds in Figure 7. As expected, the extracted graphs at initialization follow the patterns of E-R graphs (Figure 3(left)), since weight matrices are randomly i.i.d. initialized. Interestingly, after training to convergence, the extracted graphs are no longer E-R random graphs and move towards the sweet spot region we found in Section 5. Note that there is still a gap between these learned graphs and the best-performing graph imposed as a structural prior, which might explain why a fully-connected MLP has inferior performance.

In our experiments, we also find that there are a few special cases where learning the graph structure can be superior (*i.e.*, when the task is simple and the network capacity is abundant). We provide more discussions in the Appendix. Overall, these results further demonstrate that studying the graph structure of a neural network is crucial for understanding its predictive performance.

Unified view of Graph Neural Networks (GNNs) and general neural architectures. The way we define neural networks as a message exchange function over graphs is partly inspired by GNNs (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018). Under the relational graph representation, we point out that *GNNs are a special class of general neural architectures* where: (1) graph structure is regarded as the input instead of part of the neural architecture; consequently, (2) message functions are shared across all the edges to respect the invariance properties of the input graph. Concretely, recall how we define general neural networks as relational graphs:

$$\mathbf{x}_v^{(r+1)} = \text{AGG}^{(r)}(\{f_v^{(r)}(\mathbf{x}_u^{(r)}), \forall u \in N(v)\}) \quad (6)$$

For a GNN, $f_v^{(r)}$ is shared across all the edges (u, v), and $N(v)$ is provided by the input graph dataset, while a general neural architecture does not have such constraints.

Therefore, our work offers a *unified view of GNNs and general neural architecture design*, which we hope can bridge the two communities and inspire new innovations. On one hand, successful techniques in general neural architectures can be naturally introduced to the design of GNNs, such as separable convolution (Howard et al., 2017), group normalization (Wu & He, 2018) and Squeeze-and-Excitation block (Hu et al., 2018); on the other hand, novel GNN architectures (You et al., 2019b; Chen et al., 2019) beyond the commonly used paradigm (*i.e.*, Equation 6) may inspire more advanced neural architecture designs.

8. Conclusion

In sum, we propose a new perspective of using relational graph representation for analyzing and understanding neural networks. Our work suggests a new transition from studying conventional computation architecture to studying *graph structure* of neural networks. We show that well-established graph techniques and methodologies offered in other science disciplines (network science, neuroscience, *etc.*) could contribute to understanding and designing deep neural networks. We believe this could be a fruitful avenue of future research that tackles more complex situations.

Acknowledgments

This work is done during Jiaxuan You's internship at Facebook AI Research. Jure Leskovec is a Chan Zuckerberg Biohub investigator. The authors thank Alexander Kirillov, Ross Girshick, Jonathan Gomes Selman, Pan Li for their helpful discussions.

References

Ahmed, K. and Torresani, L. Maskconnect: Connectivity learning by gradient descent. In *ECCV*, 2018.

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

Alizadeh, K., Farhadi, A., and Rastegari, M. Butterfly transform: An efficient fft based neural architecture design. *arXiv preprint arXiv:1906.02256*, 2019.

Barabási, A.-L. and Pósfai, M. *Network science*. Cambridge University Press, Cambridge, 2016.

Bassett, D. S. and Bullmore, E. Small-world brain networks. *The neuroscientist*, 12(6):512–523, 2006.

Chen, Z., Villar, S., Chen, L., and Bruna, J. On the equivalence between graph isomorphism testing and function approximation with gnns. In *NeurIPS*, 2019.

Chollet, F. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, pp. 1251–1258, 2017.

Dao, T., Gu, A., Eichhorn, M., Rudra, A., and Ré, C. Learning fast algorithms for linear transforms using butterfly factorizations. In *ICML*, 2019.

Elsen, E., Dukhan, M., Gale, T., and Simonyan, K. Fast sparse convnets. *arXiv preprint arXiv:1911.09723*, 2019.

Erdős, P. and Rényi, A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Fornito, A., Zalesky, A., and Breakspear, M. Graph analysis of the human connectome: promise, progress, and pitfalls. *Neuroimage*, 80:426–444, 2013.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *AISTATS*, pp. 315–323, 2011.

Gray, S., Radford, A., and Kingma, D. P. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 2017.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

Harary, F. The maximum connectivity of a graph. *Proceedings of the National Academy of Sciences of the United States of America*, 48(7):1142, 1962.

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

Hinton, G. E. Distributed representations. 1984.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *CVPR*, 2018.

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

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Latora, V. and Marchiori, M. Efficient behavior of small-world networks. *Physical review letters*, 87(19):198701, 2001.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. In *ICML*, 2018.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *ECCV*, 2018.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *ICLR*, 2019.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

McClelland, J. L., Rumelhart, D. E., Group, P. R., et al. Parallel distributed processing. *Explorations in the Microstructure of Cognition*, 2:216–271, 1986.

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., Wang, G., Zou, Z., Wu, Z., He, W., et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Prabhu, A., Varma, G., and Namboodiri, A. Deep expander networks: Efficient deep networks from graph theory. In *ECCV*, 2018.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

Sengupta, B., Stemmler, M. B., and Friston, K. J. Information and efficiency in the nervous system—a synthesis. *PLoS computational biology*, 9(7), 2013.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Sporns, O. Graph theory methods for the analysis of neural connectivity patterns. In *Neuroscience databases*, pp. 171–185. Springer, 2003.

Stringer, C., Pachitariu, M., Steinmetz, N., Reddy, C. B., Carandini, M., and Harris, K. D. Spontaneous behaviors drive multidimensional, brain-wide population activity. *BioRxiv*, pp. 306019, 2018.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *CVPR*, 2015.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *ICLR*, 2018.

Watts, D. J. and Strogatz, S. H. Collective dynamics of 'small-world'networks. *Nature*, 393(6684):440, 1998.

Wortsman, M., Farhadi, A., and Rastegari, M. Discovering neural wirings. In *NeurIPS*, 2019.

Wu, Y. and He, K. Group normalization. In *ECCV*, 2018.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

Xie, S., Kirillov, A., Girshick, R., and He, K. Exploring randomly wired neural networks for image recognition. In *ICCV*, 2019.

Yang, J., Ren, Z., Gan, C., Zhu, H., and Parikh, D. Cross-channel communication networks. In *NeurIPS*, 2019.

Yang, Y., Zhong, Z., Shen, T., and Lin, Z. Convolutional neural networks with alternately updated clique. In *CVPR*, 2018.

Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., and Hutter, F. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. In *NeurIPS*, 2018a.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018b.

You, J., Wu, H., Barrett, C., Ramanujan, R., and Leskovec, J. G2SAT: Learning to generate sat formulas. In *NeurIPS*, 2019a.

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. *ICML*, 2019b.

Zhang, X., Zhou, X., Lin, M., and Sun, J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *ICML*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.