Crafting Efficient Neural Graph of Large Entropy

Minjing Dong\textsuperscript{1}, Hanting Chen\textsuperscript{2,3*}, Yunhe Wang\textsuperscript{3} and Chang Xu\textsuperscript{1}

\textsuperscript{1}School of Computer Science, Faculty of Engineering, University of Sydney, Australia
\textsuperscript{2}Key Laboratory of Machine Perception (Ministry of Education), Peking University, China
\textsuperscript{3}Huawei Noah’s Ark Lab

mdon0736@uni.sydney.edu.au, htchen@pku.edu.cn, yunhe.wang@huawei.com, c.xu@sydney.edu.au

Abstract

Network pruning is widely applied to deep CNN models due to their heavy computation costs and achieves high performance by keeping important weights while removing the redundancy. Pruning redundant weights directly may hurt global information flow, which suggests that an efficient sparse network should take graph properties into account. Thus, instead of paying more attention to preserving important weight, we focus on the pruned architecture itself. We propose to use graph entropy as the measurement, which shows useful properties to craft high-quality neural graphs and enables us to propose efficient algorithm to construct them as the initial network architecture. Our algorithm can be easily implemented and deployed to different popular CNN models and achieve better trade-offs.

1 Introduction

The success of Convolutional Neural Networks (CNNs) comes with massive parameters computation and storage. A wide variety of models with deeper architecture have been exploited in recent years and have achieved state-of-the-art performance in many computer vision applications, such as image classification and object detection. [Simonyan and Zisserman, 2014; He et al., 2016; Huang et al., 2016] However, due to the high computation costs and run-time memory, those deep networks cannot be directly deployed to some resource-constrained platforms, such as mobile devices and embedded sensors, which has great application potential.

Thus, reducing the storage and computation usage of deep CNN models has received increasing attention [Hassibi and Stork, 1993]. Recently, some compression algorithms have been further explored to achieve satisfactory performance in deeper and large-scale CNN model compression [Zhou et al., 2016; Yang et al., 2016; Luo et al., 2017; You et al., 2017; He et al., 2017; Yu et al., 2017; Wu et al., 2018; Bansal et al., 2018]. By pruning the neurons or channels, the network can be more sparse and efficiency of networks can be improved. [Han et al., 2015] proposes to prune the neural connections with small weights. [Li et al., 2016] proposes to prune the channels with small weights and then fine-tune the network. [Yang et al., 2016] proposes a pruning algorithm by minimizing the error in the output features. [Luo et al., 2017] prunes the channels according to the feature reconstruction error. [Yu et al., 2017] propagates the feature ranking on the final response layer to obtain neuron importance scores. [Liu et al., 2017] proposed to make use of the scaling factors in Batch normalization [Ioffe and Szegedy, 2015] for pruning channels. [Zhang et al., 2018] formulates pruning as a constrained nonconvex optimization problem.

Typical network pruning techniques focus on keeping important weights and fine-tune pruned models. However, recent works argue that the pruned architecture itself contributes to the final efficiency [Liu et al., 2018]. Getting lost in manipulating individual neurons or channels, we could ignore the big picture of the neural network. To illustrate, we constructed two toy networks of 2 layers with same number of connections under different algorithms. The random algorithm is constructed by randomly selecting the neural connections in the neural graph, whereas the regular algorithm randomly selects the neural connections under the constraint of regularity. For example, poor regularity may block data flows and hinder neurons or channels from getting involved in the network, as shown in Figure 1a, which is generated by random algorithm. It is therefore necessary to have a thorough...
investigation on the characteristics displayed by the neural network as a whole (see Figure 1b), which forces all the vertices on the same side have similar degrees.

In this paper, we propose to craft efficient deep neural network through a graph lens. Structural complexity reveals the way in which vertices and edges are arranged in the graph, providing a significant influence on the graph function and performance. Graph entropy offers an attractive route to such complexity measures. To increase the capacity of the pruned network under a particular network sparsity, we maximize graph entropy of the network by optimizing the arrangements of neurons and connections. We identify important weights from the pre-trained over-parameterized network, and use them in preference to others in crafting our efficient neural network. Based on the resulting sparse network architecture, we train the network parameters from scratch rather than adapting their original weights. The proposed algorithm can be easily deployed to many popular network architectures, such as ResNet [He et al., 2016], VGG networks [Simonyan and Zisserman, 2014] and DenseNet [Huang et al., 2016]. Experimental results on ImageNet and CIFAR datasets [Krizhevsky, 2009; Deng et al., 2009] demonstrate that deep neural networks can be well compressed by investigating graph entropy while preserving the accuracy.

2 Methodology

Pruning neural networks is to compresses networks by deleting neurons or neuron connections from a trained model, which has been paid more attention to in recent years. However, most pruning techniques only involve local operations and do not take whole network properties into consideration, which may block information flows from layer to layer. In contrast, we take neural network architecture as a graph, and construct sparse graphs with a global viewpoint to initialize the network architecture before the training phase. To encourage better information flow in the network, we employ Von Neumann Entropy as a measurement to assess the qualities of graphs, which leads to a favorable tradeoff between accuracy and sparsity of the neural network.

2.1 Von Neumann Entropy

Von Neumann Entropy is an extension of the Gibbs entropy to the quantum field, which can be treated as a quantitative measure of mixedness of density matrices [Braunstein et al., 2004]. Recently, considering its capability of describing spectral complexity, centrality, and entanglement of the graph, Von Neumann Entropy has been further explored to evaluate graph entropy in various graph pattern recognition and analysis applications. The definition of Von Neumann Entropy is given as

\[ S(\rho) = -\text{tr}(\rho \ln \rho), \] (1)

where \( \text{tr} \) denotes the trace of matrix and \( \rho \) is the density matrix. \( \rho \) could be a Laplacian matrix \( L_G \) scaled by degree sum of graph \( G \), i.e. \( \rho = \frac{1}{\text{deg}} L_G \). Given \( \lambda_i \) as the \( i \)-th eigenvalue of density matrix \( \rho \), Von Neumann Entropy can be re-written

\[ S(\rho) = -\sum_{i=1}^{n} \lambda_i \ln \lambda_i, \] (2)

The Shannon entropy computes the uncertainty of global spectral parameters of graph, involving all the eigenvalues, which makes it a useful and general measurement. Here we pay more attention to properties of Von Neumann Entropy.

To illustrate, we first constructed two toy graphs with the minimum and maximum entropy using greedy algorithm to explore its properties, as shown in Figure 2. We observed that given a fixed number of edges, if there are more connected clusters that are disjoint unions of highly fully-connected subgraph, the graph will have a smaller entropy. This is consistent with the results in [Passerini and Severini, 2008]. The entropy of the graph in Figure 2 (a) is 2.554. Almost half of connections from the first layer to bottom layer have been blocked and several vertices are deactivated due to the minimum entropy construction. On the contrary, a “balanced” graph that has a higher regularity tends to have a larger entropy. A more rigorous proof will be given later. For example, we plot a graph that is constructed with the maximum entropy given 50 edges in Figure 2 (b) whose entropy is 2.875. All vertices on the same layer have similar degree, which produces a balanced graph and results in better connections and data flows. If a graph is more balanced, every neuron would be more active in contributing to the entire neural network, which results in a better network performance.

An efficient deep neural network is expected to have a better inner connection for data flows and high-sparsity for efficient compression, a graph of large entropy exactly tickes all the boxes. Consider the neural graph \( G = (V, E) \), where \( V \) is the set of vertices with size \( n \) and \( E \) is the set of edges with size \( m \). We can use greedy algorithm to maximize the entropy of graph. We simply compute graph entropy increment by adding all the possible edges and select the one which contributes the maximum increment. Algorithm 1 shows the details of it. After the neural graph construction, a network can be easily crafted from the graph.

For a linear layer, it can easily built by treating vertices as neurons and edges as neuron connections. For a convolu-
Algorithm 1 Neural graph generation with greedy algorithm

Input: number of total edges \( m \), number of edges to select \( m' \)
Output: graph \( G \) with size of \( m' \)

Initialize graph \( G = [ ] \)
Initialize edges \( E \) with size \( m \), which are edges of complete neural graph (fully connected)
repeat
- \( \text{Entropy}_{\text{max}} = 0 \)
- for \( i = 0 \) to \( |E| - 1 \) do
  - \( \text{Entropy}(G \cup E_i) = \text{Entropy}(G \cup E_i) \)
  - if \( \text{Entropy}_{\text{max}} < \text{Entropy}(G \cup E_i) \) then
    - \( E_{\text{max}} = E_i \)
    - \( \text{Entropy}_{\text{max}} = \text{Entropy}(G \cup E_i) \)
  - else
    - Continue
  end if
end for
Add edge \( E_{\text{max}} \) to \( G \)
until \( |G| = m' \)

Algorithm 2 Random regular neural graph generation

Input, output and Initialization are same with Algorithm 1
Initialize \( D \) with the size of \( n \) to record the degrees
repeat
- For edge \( [a, b] \) in \( E \), compute the squared sum of degree
  \( dS = (d_a + 1)^2 + (d_b + 1)^2 \)
- Find the edges with minimum \( dS_{\text{min}} \) from \( E \), marked as \( E' \)
- Randomly select one edge \( [u, v] \) from \( E' \)
- Add edge \( [u, v] \) to \( G \)
- Update \( D_u \) and \( D_v \) in \( D \)
until \( |G| = m' \)

where \( d_v \) denotes the degree of vertex \( v \). Given an edge \( [a, b] \) between vertices \( a \) and \( b \), the increment in entropy of adding this edge to graph can be computed as

\[
S(\rho(G \cup [a, b])) - S(\rho(G)) \approx \frac{1}{2m} + \frac{1}{4m^2} \sum_{v \in V} d_v^2 - \frac{1}{2(m + 1)} - \frac{1}{4(m + 1)^2} (\sum_{v \neq a, b, v \in V} d_v^2 + (d_a + 1)^2 + (d_b + 1)^2). 
\]

From Eq. 7, it is obvious that the increment of entropy depends on \( d_a \) and \( d_b \). Graph \( G \) can obtain more increase in entropy if vertices \( a \) and \( b \) have smaller degrees. We reduce the cost of searching, by simply choosing the vertices pair with smallest squared sum of degrees in each step, instead of computing the entropy increment of adding all the possible edges, which reduces computation complexity to \( O(m^2) \). Algorithm 2 shows the details of our regular construction. The complexity can be further reduced by exploring the vertex degree boundaries. The objective is to minimize the squared sum of degrees and we can derive a lower bound by making use of inequality as

\[
(d_a + 1)^2 + (d_b + 1)^2 \geq \frac{(d_a + d_b + 2)^2}{2}. 
\]

From Eq. 8, we obtain a sub-optimal solution by directly minimizing the lower bound \( d_a + d_b \), which can be used to further decrease the complexity based on Proposition 1.

Proposition 1. The degrees of all vertices are in range \([\lfloor \frac{2m}{n} \rfloor, \lceil \frac{2m}{n} \rceil]\).

Proof. To prove it by contradiction, we assume the opposite. (a) If there exists a vertex \( X \) has degree of \( \lceil \frac{2m}{n} \rceil + 1 \), all other vertices have minimum degree of \( \lfloor \frac{2m}{n} \rfloor \) or \( X \) cannot be selected. And the sum degree of graph will be \( 2m + 1 \), which conflicts with the definition of graph. (b) If there exists a vertex \( X \) has degree of \( \lfloor \frac{2m}{n} \rfloor - 1 \), \( X \) has not been assigned to an edge added to the graph in the final step, which conflicts with the principle that we choose the vertices pair with smallest degree. Thus all the degrees of vertices are in range \([\lfloor \frac{2m}{n} \rfloor - 1, \lfloor \frac{2m}{n} \rfloor]\). \( \square \)

Based on proposition 1, high regularity is the property our target graph must have and we simply adopt random algorithm to generate a graph with high regularity by randomly
Figure 3: An illustration of our algorithm. Dotted lines denotes the complete neural graph and full ones denotes the edges added to the final sparse graph. With importance weights, we add edges with maximum importance scores one by one to the graph. Meanwhile, we force the regularity of graph. For example, the edges with red values are the ones with top importance scores in the layer, however, those added to graph sometimes are not exactly these edges due to the regularity.

**Algorithm 3** Regular neural graph generation with importance weights

<table>
<thead>
<tr>
<th>Input, output and Initialization are same with Algorithm 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Additional Input:</strong> importance scores $S$</td>
</tr>
<tr>
<td><strong>Sort</strong> $S$ in descending order</td>
</tr>
<tr>
<td><strong>Sort</strong> $E$ according to $S$</td>
</tr>
<tr>
<td><strong>repeat</strong></td>
</tr>
<tr>
<td>Select edge $[u, v]$ from $E$ in order</td>
</tr>
<tr>
<td>if $\text{degree}(u) &lt; U_{\text{max}<em>d}$ and $\text{degree}(v) &lt; V</em>{\text{max}_d}$ then</td>
</tr>
<tr>
<td>Add edge $[u, v]$ to $G$</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>Continue</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td><strong>until</strong> $</td>
</tr>
</tbody>
</table>

adding edges to graph while restricting degree upper boundary of all the vertices, which reduces computation complexity to $O(m)$. In terms of models which contains massive layers and neural connections, we can simply divide the entire network into several subnetworks, which further reduces the complexity of construction.

### 2.3 Importance Weight

Although we reduce the construction complexity, the output graph is not fixed due to the random algorithm which randomly select the edge with minimum squared sum of degrees because there is no criterion for selection when the graph exists multiple vertices pairs with minimum sum, which makes the performance unstable, shown in Algorithm 2. Thus, we introduce importance weights which can be easily obtained to tackle this problem by giving the selection criterion.

**Importance Estimation By Gradient**

The role of importance estimation is illustrated in Figure 3. Given the network $N$ and input $x$, the output of network is $N(x; \theta)$, where $\theta$ denotes weight parameters of network. The neuron connections (for FC layer) or filters (for CNN layer) may have different levels of importance according to the sensitivities of output to the infinitesimal changes on them. The output difference under perturbation can be estimated by simply computing the gradients of them as

$$N(x; \theta + \epsilon) - N(x; \theta) \approx \sum_{i=1}^{k} \frac{\partial(N(x; \theta))}{\partial \theta_i} \epsilon_i, \quad (9)$$

where $\epsilon$ is the perturbation on weight parameters and $k$ is the number of parameters in the network. From Eq. 9, the sensitivity depends on the gradients of learned network with respect to the weight parameters on input $x$. Thus, we can obtain estimated importance weights by computing their gradients as

$$S(\theta) = \frac{1}{M} \sum_{i=1}^{M} \frac{\partial(N(x_i; \theta))}{\partial \theta}, \quad (10)$$

where $M$ is the number of examples from dataset. From Eq. 10, importance weights can be computed by $M$ times backwards on a pre-trained model and selected dataset, which assists our initial sparse network to pay more attention to these connections with more important data flows.

Algorithm 3 shows the details of our proposed algorithm. With importance weights, we can construct the entire neural graph with high regularity by adding edges one by one according to their importance levels instead of random selection algorithm, shown in Figure 3. Thus importance weights guarantee that edges which tend to have important data flow will be added to the graph, which makes the generated network adaptive to the specific dataset so that our network has more stable performance and gains better trade-offs.

Our final algorithm, regular algorithm with importance weights (RAIW) has taken both graph entropy and importance weights into consideration, which improves the efficiency due to graph entropy and guarantees the stability due to importance weights. The entire process of crafting neural graphs is shown in Figure 3. The edges with high importance will be added to our neural graph if it does not destroy the regularity of graph. For example, the edge with the top importance weights 0.01 cannot be added to our neural graph because it connects those vertices with higher degrees, thus the edge with value of 0.005 is added instead, shown in the final step in Figure 3.
3 Experiments
To evaluate the efficiency of our algorithms, we apply our RAIW algorithm to generate neural graphs based on different popular CNN architectures, such as VGG, Resnet and Densenet. For comparison, we repeat the experiments on different datasets, such as CIFAR10, CIFAR100 and Imagenet, with these architectures under various layer settings and compare them with pruning techniques or the original models to demonstrate better trade-offs of our algorithm. The stability and regularity will also be discussed.

3.1 Comparison With Efficient CNN Architectures and Pruning Algorithms

VGG on CIFAR10
We compare our algorithm against some popular pruning techniques, [Han et al., 2015; Li et al., 2016; Liu et al., 2017] [Liu et al., 2017] which achieve good performance among pruning techniques. To evaluate the performance of RAIW, we evaluate on CIFAR10 [Krizhevsky, 2009] with VGG-16 architecture. The detailed results are shown in Table 1. Our algorithm can preserve the accuracy, which has 0.6% drop but only 1.1M parameters, almost 14X compression rate.

Densenet and Resnet on CIFAR100
To evaluate the robustness of our algorithm, we deploy RAIW on Densenet and Resnet running on CIFAR100 dataset [Krizhevsky, 2009]. We sparse these models by crafting convolutional layers whose filter size is half of the original one by controlling the number of selected edges in algorithms.

For Resnet on CIFAR100, we run Resnet with different number of layers on CIFAR100 dataset. We compare our algorithm with these base models by comparing the 1-crop error along with the number of parameters of model, the details are given in Figure 4 (a). Our algorithm consistently has a better performance with the similar parameters, comparing the two lines in Figure 4 (a). For example, the original Resnet-56 has 0.86M parameters number with 28.89% error, however, our RAIW algorithm which has the similar parameter number on Resnet-110, has 0.8% error drop.

For Densenet on CIFAR100, we run with Densenet-BC which contains bottleneck layers and uses Densenet-BC-40-24, 40-48, 40-60 which have 40 layers and different growth rates as base models. Again, we show better accuracy-parameters trade-offs, the details are given in Figure 4 (b). Similarly, comparing the two lines in Figure 4 (b), the network we crafted using RAIW algorithm can be more efficient. For example, RAIW algorithm has 21.07% error on Densenet-BC-40-60 with 2.10M parameters while the original Densenet-BC-40-48 has 21.37% error with 2.76M parameters, which demonstrates the efficiency of our algorithm.

Table 2: The accuracy performance of Resnet crafted by RAIW evaluated on Imagenet dataset, compared with original architectures, ordered by number of parameters.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAIW-Resnet-50</td>
<td>70.4%</td>
<td>13.28M</td>
</tr>
<tr>
<td>RAIW-Resnet-101</td>
<td>73.6%</td>
<td>21.08M</td>
</tr>
<tr>
<td>Resnet-34</td>
<td>73.3%</td>
<td>21.78M</td>
</tr>
<tr>
<td>Resnet-50</td>
<td>75.3%</td>
<td>25.50M</td>
</tr>
</tbody>
</table>

Table 3: VGG16 model under construction of regular algorithm and the one with importance weight over 5 runnings on CIFAR10 dataset. The final column “Average” denotes the mean accuracy ± standard deviation.

<table>
<thead>
<tr>
<th>ALGO</th>
<th>R1%</th>
<th>R2%</th>
<th>R3%</th>
<th>R4%</th>
<th>R5%</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>92.69</td>
<td>92.66</td>
<td>92.97</td>
<td>92.70</td>
<td>93.06</td>
<td>92.82±0.17</td>
</tr>
<tr>
<td>RAIW</td>
<td>92.97</td>
<td>92.77</td>
<td>93.00</td>
<td>92.82</td>
<td>92.69</td>
<td>92.85±0.12</td>
</tr>
</tbody>
</table>

Table 1: The performance of VGG16 network crafted by RAIW algorithm compared with original VGG16 and pruning techniques on CIFAR 10 dataset.
Figure 4: Trade-offs between number of parameters and error shown in (a) and (b). The orange lines denote error-parameters number trade-offs of our algorithm and the blue ones denote the original architecture. The two-line numbers on each data point in (a) denote the number of layers in ResNet on the first line and the corresponding error on the second line. Those in (b) denote the number of layers in DenseNet with its growth rate on the first line and the corresponding error on the second line. We show the performance of our algorithm applied to Resnet on CIFAR100 in (a), Densenet on CIFAR100 in (b). (c) illustrates entropy variation of 256*256 and 512*512 layers construction with degree of 16. From the line chart, regular algorithm can guarantee much faster entropy increment compared with random algorithm.

<table>
<thead>
<tr>
<th>VGG16</th>
<th>Original VGG</th>
<th>d-32 VGG</th>
<th>d-16 VGG</th>
<th>d-8 VGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv1</td>
<td>3x64,3x3,3</td>
<td>3x64,3x3</td>
<td>3x64,3x3</td>
<td>3x64,3x3</td>
</tr>
<tr>
<td>Conv2</td>
<td>6x64,3x3,3</td>
<td>6x64,3x3</td>
<td>6x64,3x3</td>
<td>6x64,3x3</td>
</tr>
<tr>
<td>Conv3</td>
<td>6x128,3x3</td>
<td>6x128,3x3</td>
<td>6x128,3x3</td>
<td>6x128,3x3</td>
</tr>
<tr>
<td>Conv4</td>
<td>128x128,3x3</td>
<td>128x128,3x3</td>
<td>128x128,3x3</td>
<td>128x128,3x3</td>
</tr>
<tr>
<td>Conv5</td>
<td>128x256,3x3</td>
<td>128x256,3x3</td>
<td>128x256,3x3</td>
<td>128x256,3x3</td>
</tr>
<tr>
<td>Conv6</td>
<td>256x256,3x3</td>
<td>256x256,3x3</td>
<td>256x256,3x3</td>
<td>256x256,3x3</td>
</tr>
<tr>
<td>Conv7</td>
<td>256x512,3x3</td>
<td>256x512,3x3</td>
<td>256x512,3x3</td>
<td>256x512,3x3</td>
</tr>
<tr>
<td>Conv8</td>
<td>512x512,3x3</td>
<td>512x512,3x3</td>
<td>512x512,3x3</td>
<td>512x512,3x3</td>
</tr>
<tr>
<td>Conv9</td>
<td>512x1024,3x3</td>
<td>512x1024,3x3</td>
<td>512x1024,3x3</td>
<td>512x1024,3x3</td>
</tr>
<tr>
<td>Linear1</td>
<td>512x10</td>
<td>512x10</td>
<td>512x10</td>
<td>512x10</td>
</tr>
<tr>
<td>Linear2</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>Total</td>
<td>14.98M</td>
<td>1.07M</td>
<td>0.75M</td>
<td>0.55M</td>
</tr>
<tr>
<td>Accuracy</td>
<td>93.96%</td>
<td>93.37%</td>
<td>93.06%</td>
<td>91.85%</td>
</tr>
</tbody>
</table>

Table 4: Details of each layer and number of parameters with accuracy of VGG16 model under different sparsity.

This work was supported in part by the Australian Research Council under Project DE180101438.
References


