You Can Have Better Graph Neural Networks by Not Training Weights at All: Finding Untrained GNNs Tickets

Anonymous Author(s)
Anonymous Affiliation
Anonymous Email

Abstract

Recent works have impressively demonstrated that there exists a subnetwork in randomly initialized convolutional neural networks (CNNs) that can match the performance of the fully trained dense networks at initialization, without any optimization of the weights of the network (i.e., untrained networks). However, the presence of such untrained subnetworks in graph neural networks (GNNs) still remains mysterious. In this paper we carry out the first-of-its-kind exploration of discovering matching untrained GNNs. With sparsity as the core tool, we can find untrained sparse subnetworks at the initialization, that can match the performance of fully trained dense GNNs. Besides this already encouraging finding of comparable performance, we show that the found untrained subnetworks can substantially mitigate the GNN over-smoothing problem, hence becoming a powerful tool to enable deeper GNNs without bells and whistles. We also observe that such sparse untrained subnetworks have appealing performance in out-of-distribution detection and robustness of input perturbations. We evaluate our method across widely-used GNN architectures on various popular datasets including the Open Graph Benchmark (OGB). Our source codes are submitted in the Supplementary.

1 Introduction

Graph Neural Networks (GNNs) [1, 2] have shown the power to learn representations from graph-structured data. Over the past decade, GNNs and their variants such as Graph Convolutional Networks (GCN) [3], Graph Isomorphism Networks (GIN) [4], Graph Attention Networks (GAT) [5] have been successfully applied to a wide range of scenarios, e.g., social analysis [6, 7], protein feature learning [8], traffic prediction [9], and recommendation systems [10]. In parallel, works on untrained networks [11, 12] surprisingly discover the presence of untrained subnetworks in CNNs that can already match the accuracy of their fully trained dense CNNs with their initial weights, without any weight update. In this paper, we attempt to explore discovering untrained sparse networks in GNNs by asking the following question:

Is it possible to find a well-performing graph neural (sub-) network without any training of the model weights?

Positive answers to this question will have significant impacts on the research field of GNNs. ① If the answer is yes, it will shed light on a new direction of obtaining performant GNNs, e.g., traditional training might not be indispensable towards performant GNNs. ② The existence of such performant subnetworks will extend the recently proposed untrained subnetwork techniques [11, 12] in GNNs. Prior works [11–13] successfully find that randomly weighted full networks contain untrained subnetworks which perform well without ever modifying the weights, in convolutional neural networks (CNNs). However, the similar study has never been discussed for GNNs. While CNNs reasonably contain well-performing untrained subnetworks due to heavy over-parameterization, GNN models are usually much more compact, and it is unclear whether a performant subnetwork “should” still exist in GNNs.

Submitted to the First Learning on Graphs Conference (LoG 2022). Do not distribute.
2 Related Work

Graph Neural Networks. Graph neural networks is a powerful deep learning approach for graph-structured data. Since proposed in [1], many variants of GNNs have been developed, e.g., GAT [5], GCN [3], GIN [4], GraphSage [21], SGC [22], and GAE [23]. More and more recent works point out that deeper GNN architectures potentially provide benefits to practical graph structures, e.g., molecules [8], point clouds [24], and meshes [25], as well as large-scale graph dataset OGB. However, training deep GNNs usually is a well-known challenge due to various difficulties such as gradient vanishing and over-smoothing problems [14, 26]. The existing approaches to address the above-mentioned problem can be categorized into three groups: (1) skip connection, e.g., Jumping connections [27, 28], Residual connections [24], and Initial connections [29]; (2) graph normalization, e.g., PairNorm [26], NodeNorm [30]; (3) random dropping including DROPNode [31] and DROPEdge [32].

Untrained Subnetworks. Untrained subnetworks refer to the hypothesis that there exists a subnetwork in a randomly initialized neural network that can achieve almost the same accuracy as a fully trained neural network without weight update. [11] and [12] first demonstrate that randomly
initialized CNNs contain subnetworks that achieve impressive performance without updating weights at all. [13] enhanced the performance of untrained subnetworks by iteratively reinitializing the weights that have been pruned. Besides the image classification task, some works also explore the power of untrained subnetworks in other domains, such as multi-tasks learning [33] and adversarial robustness [34].

Instead of proposing well-versed techniques to enable deep GNNs training, we explore the possibility of finding well-performing deeper graph subnetworks at initialization in the hope of avoiding the difficulties of building deep GNNs without model weight training.

3 Untrained GNNs Tickets

3.1 Preliminaries and Setups

Notations. We represent matrices by bold uppercase characters, e.g. $X$, vectors by bold lowercase characters, e.g. $x$, and scalars by normal lowercase characters, e.g. $x$. We denote the $i^{th}$ row of a matrix $A$ by $A[i, :]$, and the $(i, j)^{th}$ element of matrix $A$ by $A[i, j]$. We consider a graph $G = \{V, E\}$ where $E$ is a set of edges and $V$ is a set of nodes. Let $g(A, X; \theta)$ be a graph neural network where $A \in \{0, 1\}^{|V| \times |V|}$ is adjacency matrix for describing the overall graph topology, and $X$ denotes nodal features. $A[i, j] = 1$ denotes the edge between node $v_i$ and node $v_j$. Let $f(X; \theta)$ be a neural network with the weights $\theta$. $\| \cdot \|_0$ denotes the $L_0$ norm.

Sparse Neural Networks. Given a dense network $\theta_l \in \mathcal{R}^{d_l}$ with a dimension of $d_l$ in each layer $l \in \{1, ..., L\}$, binary mask $m_l \in \{0, 1\}^{d_l}$ yielding a sparse Neural Networks with sparse weights $\theta_l \odot m_l$. The sparsity level is the fraction of the weights that are zero-valued, calculated as $s = 1 - \frac{\sum ||m_l||_0}{\sum_{i} d_i}$.

Graph Neural Networks. GNNs denote a family of algorithms that extract structural information from graphs [35] and it is consisted of Aggregate and Combine operations. Usually, Aggregate is a function that aggregates messages from its neighbor nodes, and Combine is an update function that updates the representation of the current node. Formally, given the graph $G = (A, X)$ with node set $V$ and edge set $E$, the $l$-th layer of a GNN is represented as follows:

$$a^l_v = Aggregate^l(\{h^{l-1}_u : \forall u \in N(v)\})$$

$$h^l_v = Combine^l(h^{l-1}_v, a^l_v)$$

where $a^l_v$ is the aggregated representation of the neighborhood for node $v$ and $N(v)$ denotes the neighbor nodes set of the node $v$, and $h^l_v$ is the node representations at the $l$-th layer. After propagating through $L$ layers, we achieve the final node representations $h^L_v$ which can be applied to downstream node-level tasks, such as node classification, link prediction.

Untrained Subnetworks. Following the prior work [11], [12] proposed Edge-Popup which enables finding untrained subnetworks hidden in the a randomly initialized full network $f(\theta)$ by solving the following discrete optimization problem:

$$\min_{m(s) \in \{0, 1\}^{|\theta|}} \mathcal{L}(f(X; \theta \odot m(S)), y)$$

where $\mathcal{L}$ is task-dependent loss function; $\odot$ represents an element-wise multiplication; $y$ is the label for the input $X$ and $m$ is the binary mask that controls the sparsity level $s$. $S$ is the latent score behind the binary mask $m$ and it has the same dimension as $m$. To avoid confusion, here we use $m(S)$ instead of $m$ to indicate that $m$ is generated by $S$. We will use $m$ directly for brevity in the following content.

Different from the traditional training of deep neural networks, here the network weights are never updated, masks $m$ are instead generated to search for the optimal untrained subnetwork. In practice, each mask $m_l$ has a latent score variable $S_l \in \mathcal{R}$ that represents the importance score of the corresponding weight $\theta_l$. During training in the forward pass, the binary mask $m_l$ is generated by setting top-$s$ smallest elements of $S_l$ to 0 otherwise 1. In the backward pass, all the values in $S_l$ will be updated with straight-through estimation [36]. At the end of the training, an untrained subnetwork can be found by the generated mask $m$ according to the converged scores $S$.
3.2 Untrained GNNs Tickets – UGTs

In this section, we adopt the untrained subnetwork techniques to GNNs and introduce our new approach – Untrained GNNs Tickets (UGTs). We share the pseudocode of UGTs in the Appendix C.

Formally, given a graph neural network $g(A, X; \theta)$, where $A$ and $X$ are adjacency matrix and nodal features respectively. The optimization problem of finding an untrained subnetwork in GNNs can be therefore described as follows:

$$\min_{m \in \{0, 1\}^{|\theta|}} \mathcal{L}(g(A, X; \theta \odot m), y)$$

(4)

Although Edge-Popup [12] can find untrained subnetworks with proper predictive accuracy, its performance is still away from satisfactory. For instance, Edge-Popup can only obtain matching subnetworks at a relatively low sparsity i.e., 50%.

We highlight two limitations of the existing prior research. First of all, prior works [12, 13] initially set the sparsity level of $m_i$ as $s$ and maintain it throughout the optimization process. This is very appealing for the scenarios of sparse training [37–39] that chases a better trade-off between performance and efficiency, since the fixed sparsity usually translates to fewer floating-point operations (FLOPs). This scheme, however, is not necessary and perhaps harmful to the finding of the smallest possible untrained subnetwork that still performs well. Particularly as shown in [20], larger searching space for sparse neural networks at the early optimization phase leads to better sparse solutions. The second limitation is that the existing methods sparsity networks layer-wise with a uniform sparsity ratio, which typically leads to inferior performance compared with the non-uniform layer-wise sparsity [20, 39, 40], especially for deep architectures [41].

Untrained GNNs Tickets (UGTs). Leveraging the above-mentioned insights, we propose a new approach UGTs here which can discover matching untrained subnetworks with extremely high sparsity levels, i.e., up to 99%. Instead of keeping the sparsity of $m$ fixed throughout the sparsification process, we start from an untrained dense GNNs and gradually increase the sparsity to the target sparsity during the whole sparsification process. We adjust the original gradual sparsification schedule [18, 19] to the linear decay schedule, since no big performance difference can be observed. The sparsity level $s_t$
Table 1: Test accuracy (%) of different training techniques. The experiments are based on GCN models with 16, 32 layers, respectively. Width is set to 448. See Appendix B.7 for GAT architecture. The results of the other methods are obtained from [42].

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>Citeese</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-Layers</td>
<td>16</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>Trained Dense GCN</td>
<td>21.4</td>
<td>21.2</td>
<td>19.5</td>
</tr>
<tr>
<td>+Residual</td>
<td>20.1</td>
<td>19.6</td>
<td>20.8</td>
</tr>
<tr>
<td>+Jumping</td>
<td>76.0</td>
<td>75.5</td>
<td>58.3</td>
</tr>
<tr>
<td>+NodeNorm</td>
<td>21.5</td>
<td>21.4</td>
<td>18.8</td>
</tr>
<tr>
<td>+PairNorm</td>
<td>55.7</td>
<td>17.7</td>
<td>27.4</td>
</tr>
<tr>
<td>+DropNode</td>
<td>27.6</td>
<td>27.6</td>
<td>21.8</td>
</tr>
<tr>
<td>+DropEdge</td>
<td>28.0</td>
<td>27.8</td>
<td>22.9</td>
</tr>
<tr>
<td>UGTs-GCN</td>
<td>77.3 ± 0.9</td>
<td>77.5 ± 0.8</td>
<td>61.1 ± 0.9</td>
</tr>
</tbody>
</table>

of each adjusting step $t$ is calculated as follows:

$$s_t = s_f + (s_i - s_f) \left(1 - \frac{t - t_0}{n\Delta t}\right)$$

(5)

where $s_f$ and $s_i$ refer to the final sparsity and initial sparsity, respectively; the initial sparsity is the sparsity at the start point of sparsification and it is set to 0 in this study. The final sparsity is the sparsity at the endpoint of sparsification. $t_0$ is the starting point of sparsification; $\Delta t$ is the time between two adjusting steps; $n$ is the total number of adjusting steps. We set $\Delta t$ as one epoch of mask optimization in this paper.

To obtain a good non-uniform layer-wise sparsity ratio, we remove the weights with the smallest score values ($S$) across layers at each adjusting step. We do this because [20] showed that the layer-wise sparsity obtained by this scheme outperforms the other well-studied sparsity ratios [19, 37, 39]. More importantly, removing weights across layers theoretically has a larger search space than solely considering one layer. The former can be more appealing as the GNN architecture goes deeper.

4 Experimental Results

In this section, we conduct extensive experiments among multiple GNN architectures and datasets to evaluate UGTs. We summarize the experimental setups here.

Table 2: Graph datasets statistics.

<table>
<thead>
<tr>
<th>DataSets</th>
<th>#Graphs</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>#Classes</th>
<th>#Features</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>1</td>
<td>2708</td>
<td>5429</td>
<td>7</td>
<td>1433</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Citeseer</td>
<td>1</td>
<td>3327</td>
<td>4732</td>
<td>6</td>
<td>3703</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Pubmed</td>
<td>1</td>
<td>19717</td>
<td>44338</td>
<td>3</td>
<td>3288</td>
<td>Accuracy</td>
</tr>
<tr>
<td>OGBN-Arxiv</td>
<td>1</td>
<td>169343</td>
<td>1166243</td>
<td>40</td>
<td>128</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Texas</td>
<td>1</td>
<td>183</td>
<td>309</td>
<td>5</td>
<td>1703</td>
<td>Accuracy</td>
</tr>
<tr>
<td>OGBN-Products</td>
<td>1</td>
<td>24449029</td>
<td>61859140</td>
<td>47</td>
<td>100</td>
<td>Accuracy</td>
</tr>
<tr>
<td>OGBG-molhiv</td>
<td>41127</td>
<td>25.5(Average)</td>
<td>27.5(Average)</td>
<td>2</td>
<td>-</td>
<td>ROC-AUC</td>
</tr>
<tr>
<td>OGBG-molbace</td>
<td>1513</td>
<td>34.1(Average)</td>
<td>36.9(Average)</td>
<td>2</td>
<td>-</td>
<td>ROC-AUC</td>
</tr>
</tbody>
</table>

GNN Architectures. We use the three most widely used GNN architectures: GCN, GIN, and GAT \(^1\) in our paper.

Datasets. We choose three popular small-scale graph datasets including Cora, Citeseer, PubMed [3] and one latest large-scale graph dataset OGBN-Arxiv [15] for our main experiments. To draw a solid conclusion, we also evaluate our method on other datasets including OGBN-Products [15], Texas [43], OGBG-molhiv [15] and OGBG-molbace [15, 44]. More detailed information can be found in Table 2.

\(^1\)All experiments based on GAT architecture are conducted with heads=1 in this study.
4.1 The Existence of Matching Subnetworks

Figure 1 shows the effectiveness of UGTs with different GNNs, including GCN, GIN and GAT, on the four datasets. We can observe that as the model size increases, UGTs can find untrained subnetworks that match the fully-trained dense GNNs. This observation is perfectly in line with the previous findings [12, 13], which reveal that model size plays a crucial role to the existence of matching untrained subnetworks. Besides, it can be observed that the proposed UGTs consistently outperforms Edge-Popup across different settings.

![Figure 1: Efficacy of UGTs with different GNNs, including GCN, GIN and GAT, on the four datasets.](image1.png)

4.2 Over-smoothing Analysis

Deep architecture has been shown as a key factor that improves the model capability in computer vision [45]. However, it becomes less appealing in GNNs mainly because the node interaction through the message-passing mechanism (i.e., aggregation operator) would make node representations less distinguishable [26, 46], leading to a drastic drop of task performance. This phenomenon is well known as the over-smoothing problem [14, 42]. In this paper, we show a surprising result that UGTs can effectively mitigate over-smoothing in deep GNNs. We conduct extensive experiments to evaluate this claim in this section.

UGTs preserves the high accuracy as GNNs go deeper. In Figure 2, we vary the model depth of various architectures and report the test accuracy. All the experiments are conducted with architectures containing width 448 except for GAT on OGBN-Arxiv, in which we choose width 256 for GAT with 2 ~ 10 layers and width 128 for GAT with 11 ~ 20 layers, due to the memory limitation.

As we can see, the performance of trained dense GNNs suffers from a sharp performance drop when the model goes deeper, whereas UGTs impressively preserves the high accuracy across models. Especially at the mild sparsity, i.e., 0.1, UGTs almost has no deterioration with the increased number of layers.

UGTs achieves competitive performance with the well-versed training techniques. To further validate the effectiveness of UGTs in mitigating over-smoothing, we compare UGTs with six state-of-the-art techniques for the over-smoothing problem, including Residual connections, Jumping connections, NodeNorm, PairNorm, DropEdge, and DropNode. We follow the experimental setting in [42] and conduct experiments on Cora/Citeseer/Pubmed with GAT containing 16 and 32 layers. Model width is set to 448 for GAT on Cora/Citeseer/Pubmed. The results of the other methods are obtained from [42].

Table 1 shows that UGTs consistently outperforms all these advanced techniques on Cora, Citeseer, and Pubmed. For instance, UGTs outperforms the best performing technique (+Jumping) by 2.0%, 1.2%, 1.0% on Cora, Citeseer and Pubmed respectively with 32 layers. These results again verify our

![Figure 3: TSNE visualization of node representations learned by densely trained GCN and UGTs. Ten classes are randomly sampled from OGBN-Arxiv for visualization. Model depth is set as 16 and 32 respectively; width is set as 448. See Appendix B.1 for GAT architecture.](image2.png)

---

Table 1 shows that UGTs consistently outperforms all these advanced techniques on Cora, Citeseer, and Pubmed. For instance, UGTs outperforms the best performing technique (+Jumping) by 2.0%, 1.2%, 1.0% on Cora, Citeseer and Pubmed respectively with 32 layers. These results again verify our

---

hypothesis that training bottlenecks of deep GNNs (e.g., over-smoothing) can be avoided or mitigated by finding untrained subnetworks without training weights at all.

Figure 4: Mean Average Distance among node representations of each GNN layer. Experiments are conducted on Cora with GCN containing 32 layers and width 448.

Figure 5: The accuracy of GNNs w.r.t varying sparsities. Experiments are conducted on various GNNs with 2 layers and width 256 for Cora, Citeseer and Pubmed, 4 layers and width 386 for OGBN-Arxiv.

Mean Average Distance (MAD). To further evaluate whether or not the good performance of UGTs can be contributed to the mitigation of over-smoothing, we visualize the smoothness of the node representations learned by UGTs and trained dense GNNs respectively. Following [46], we calculate the MAD distance among node representations for each layer during the process of sparsification. Concretely, MAD [46] is the quantitative metric for measuring the smoothness of the node representations. The smaller the MAD is, the smoother the node representations are. Results are reported in Figure 4. It can be observed that the node representations learned by UGTs keeps having a large distance throughout the optimization process, indicating a relieving of over-smoothing. On the contrary, the densely trained GCN suffers from severely indistinguishable representations of nodes.

TSNE Visualizations. Additionally, we visualize the node representations learned by UGTs and the trained dense GNNs with 16 and 32 layers, respectively, on both GCN and GAT architectures. Due to the limited space, we show the results of GCN in Figure 3 and put the visualization of GAT in the Appendix B.1. We can see that the node representations learned by the trained dense GCN are over-mixing in all scenarios and, in the deeper models (i.e., 32 layers), seem to be more indistinguishable. Meanwhile, the projection of node representations learned by UGTs maintains clearly distinguishable, again providing the empirical evidence of UGTs in mitigating over-smoothing problem.

4.3 The Effect of Sparsity on UGTs

To better understand the effect of sparsity on the performance of UGTs, we provide a comprehensive study in Figure 5 where the performance of UGTs with respect to different sparsity levels on different architectures. We summarize our observations below.

1. UGTs consistently finds matching untrained graph subnetworks at a large range of sparsities, including the extreme ones. A matching untrained graph subnetwork can be identified with sparsities...
from 0.1 even up to 0.99 on small-scale datasets such as Cora, Citeseer and Pubmed. For large-scale OGBN-Arxiv, it is more difficult to find matching untrained subnetworks. Matching subnetworks are mainly located within sparsities of $0.3 \sim 0.6$.

What’s more, UGTs consistently outperforms Edge-Popup. UGTs shows better performance than Edge-Popup at high sparsities across different architectures on Cora, Citeseer, Pubmed and OGBN-Arxiv. Surprisingly, increasing sparsity from 0.7 to 0.99, UGTs maintains very a high accuracy, whereas the accuracy of Edge-Popup shows a notable degradation. It is in accord with our expectation since UGTs finds important weights globally by searching for the well-performing sparse topology across layers.

4.4 Broader Evaluation of UGTs

![Figure 6: Out-of-distribution performance (ROC-AUC). Experiments are conducted with GCN (Width: 256, Depth: 2).](image)

![Figure 7: The robust performance on feature perturbations with the fraction of perturbed nodes varying from 0% to 40%. Experiments are conducted with GCN and GAT (Width: 256, Depth: 2).](image)

In this section, we systematically study the performance of UGTs on out of distribution (OOD) detection, robustness against the input perturbations including feature and edge perturbations. Following [47], we create OOD samples by specifying all samples from 40% of classes and removing them from the training set. We create feature perturbations by replacing them with the noise sampled from Bernoulli distribution with $p=0.5$ and edge perturbations by moving edge’s end point at random. The results of OOD experiments are reported in Figure 6 and Figure 10 (shown in Appendix B.2). The results of robustness experiments are reported in Figure 8 and Figure 7. We summarize our observations as follows:

1. UGTs enjoys matching performance on OOD detection. Figure 6 and Figure 10 show that untrained graph subnetworks discovered by UGTs achieve matching performance on OOD detection compared with the trained dense GNNs in most cases. Besides, UGTs consistently outperforms Edge-Popup method at a large range of sparsities on OOD detection.
UGTs produces highly sparse yet robust subnetworks on input perturbations. Figure 7 and Figure 8 demonstrate that UGTs with high sparsity level (Sparsity=0.9) achieves more robust results than the trained dense GNNs on both feature and edge perturbations with perturbation percentage ranging from 0 to 40%. Again, UGTs consistently outperforms Edge-Popup with both perturbation types.

4.5 Experiments on Graph-level Task and Other Datasets

To draw a solid conclusion, we further conduct extensive experiments of graph-level task on OGBG-molhiv and OGBG-molbace; node-level task on TEXAS and OGBN-Products. The experiments are based on GCN model with width=448 and depth=3. Table 3 consistently verifies that a matching untrained subnetwork can be identified in GNNs across multiple tasks and datasets.

Table 3: Experiments on graph-level tasks and other datasets. GCN Model with width:448, depth:3 are adopted for this experiments.

<table>
<thead>
<tr>
<th>Sparsity=</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>0.95</th>
<th>0.99</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
<td>79.5</td>
</tr>
<tr>
<td>UGTs</td>
<td>75.6</td>
<td>77.7</td>
<td>78.7</td>
<td>77.8</td>
<td>79.3</td>
<td>79.5</td>
<td>79.9</td>
<td>78.6</td>
<td>74.5</td>
<td>64.1</td>
<td>35.3</td>
</tr>
</tbody>
</table>

Node-level: OGBN-Products (Accuracy:%)

| Dense     | 62.2| 62.2| 62.2| 62.2| 62.2| 62.2| 62.2| 62.2| 62.2| 62.2| 62.2|
| UGTs      | 62.1| 62.2| 62.2| 62.2| 62.2| 62.2| 62.1| 63.1| 64.8| 64.8| 55.8|

Node-level: TEXAS (Accuracy:%)

| Dense     | 77.4| 77.4| 77.4| 77.4| 77.4| 77.4| 77.4| 77.4| 77.4| 77.4| 77.4|
| UGTs      | 76.4| 76.5| 76.5| 76.1| 76.3| 77.3| 75.8| 77.1| 73.1| 75.3| 75.1|

Graph-level: OGBG-molhiv (ROCAUC:%)

| Dense     | 78.3| 78.3| 78.3| 78.3| 78.3| 78.3| 78.3| 78.3| 78.3| 78.3| 78.3|
| UGTs      | 76.0| 73.7| 77.0| 77.1| 77.0| 77.6| 78.4| 77.3| 76.2| 75.6| 74.9|

Graph-level: OGBG-molbace (ROCAUC:%)

5 Conclusion

In this work, we for the first time confirm the existence of matching untrained subnetworks at a large range of sparsity. UGTs consistently outperforms the previous untrained technique – Edge-Popup on multiple graph datasets across various GNN architectures. What’s more, we show a surprising result that searching for an untrained subnetwork within a randomly weighted dense GNN instead of directly training the latter can significantly mitigate the over-smoothing problem of deep GNNs. Across popular datasets, e.g., Cora, Citeseer, Pubmed, and OGBN-Arxiv, our method UGTs can achieve comparable or better performance with the various well-studied techniques that are specifically designed for over-smoothing. Moreover, we empirically find that UGTs also achieves appealing performance on other desirable aspects, such as out-of-distribution detection and robustness. The strong results of our paper point out a surprising but perhaps worth-a-try direction to obtain high-performing GNNs, i.e., finding the Untrained Tickets located within a randomly weighted dense GNN instead of training it.
References


A Implementation Details

In this paper, all experiments on Cora/Citeseer/Pubmed datasets are conducted on 1 GeForce RTX 2080Ti (11GB) and all experiments on OGBN-Arxiv are conducted on 1 DGX-A100 (40GB). All the results reported in this paper are conducted by 5 independent repeated runs.

Train-Val-Test splitting Datasets We use 140 (Cora), 120 (Citeseer) and 60 (PubMed) labeled data for training, 500 nodes for validation and 1000 nodes for testing. We follow the strategy in [15] for splitting OGBN-Arxiv dataset.

Hyper-parameter Configuration We follow [3, 42, 48] to configure the hyper-parameters for training dense GNN models. All hyper-parameters configurations for UGTs are summarized in Table 4.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
<th>OGBN-Arxiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Epochs</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01 (GNNs with Layers&lt;10)</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
<td>Adam</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>n t (total adjustment epoches)</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>δ (epoch)</td>
<td>1 epoch</td>
<td>1 epoch</td>
<td>1 epoch</td>
<td>1 epoch</td>
</tr>
</tbody>
</table>

Table 4: Implementation details for UGTs.

B More Experimental Results

B.1 TSNE visualization.

Figure 9 provides the TSNE visualization for node representations learned by UGTs and dense GAT. It can be observed that the node representations learned by the trained dense GAT are mixed while the node representations learned by UGTs are disentangled.

Figure 9: TSNE visualization for node representations. Experiments are based on GAT with fixed width 448.

B.2 Out of distribution detection

Figure 10 shows the OOD performance for UGTs and the trained dense GNNs based on GAT architecture. As we can observe, UGTs achieves very appealing results on OOD performance than the corresponding trained dense GAT.

B.3 Robustness against input perturbations

In this section, we explore the robustness against input perturbations with varying the sparsity of untrained GNNs. Experiments are conducted on GAT and GCN architectures with width=256 and depth=2. Results are reported in Figure 12 and Figure 11.
Figure 10: Out-of-distribution performance (ROC-AUC). Experiments are based on GAT architecture (Width:256, Depth:2)

It can be observed that the robustness achieved by UGTs is increasing with the increase of sparsity for both edge and feature perturbation types. Besides, the robustness achieved by UGTs at large sparsity, e.g., sparsity =0.9, can outperform the counterpart trained dense GNNs.

Figure 11: The robust performance on edge perturbations. R denotes the fraction of perturbed edges. (width:256, Depth:2)

Figure 12: The robust performance on feature perturbations. R denotes the fraction of perturbed nodes. (width:256, Depth:2)
B.4 The accuracy performance w.r.t model width

Figure 13 shows the performance of UGTs on different architectures with varying model width from 16 to 1024 and fix depth=2. We summarize observations as follows:

Performance of UGTs improves with the width of the GNN models. With width increasing from 16 to 256, the performance of UGTs improves apparently and after width=256, the benefits from model width are saturated.

B.5 Ablation studies

We conduct the ablation studies to show the effectiveness of UGTs. The results showed on Table 5. Compared with Edge-Popup, UGTs mainly has two novelties: global sparsification (VS. uniform sparsification) and gradual sparsification (VS. one-shot sparsification). Here we compare UGTs with 3 baselines: (1) UGTs - global sparsification; (2) UGTs - gradual sparsification; (3) Edge-Popup. The results are reported in the following table and it shows that global sparsification plays an important role for finding important weights and gradual sparsification is crucial for further boosting performance at high sparsity level.

Table 5: Ablation studies based on GAT (Depth:2, Width:256) and Cora.

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.3</th>
<th>0.5</th>
<th>0.7</th>
<th>0.9</th>
<th>0.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge-Popup</td>
<td>0.814</td>
<td>0.81</td>
<td>0.809</td>
<td>0.81</td>
<td>0.791</td>
<td>0.461</td>
</tr>
<tr>
<td>UGTs - global sparsification</td>
<td>0.807</td>
<td>0.816</td>
<td>0.817</td>
<td>0.804</td>
<td>0.799</td>
<td>0.731</td>
</tr>
<tr>
<td>UGTs - gradual sparsification</td>
<td>0.806</td>
<td><strong>0.818</strong></td>
<td><strong>0.821</strong></td>
<td><strong>0.821</strong></td>
<td>0.804</td>
<td>0.795</td>
</tr>
<tr>
<td>UGTs</td>
<td>0.797</td>
<td>0.811</td>
<td>0.81</td>
<td>0.815</td>
<td><strong>0.817</strong></td>
<td><strong>0.822</strong></td>
</tr>
</tbody>
</table>

B.6 Observations via gradient norm

To preliminary understand why UGTs can mitigate over-smoothing while the trained dense GNNs cannot, we calculate the gradient norm of each layer for UGTs and dense GCN during training. In order to have a fair comparison, we calculate the gradient norm of $\nabla (m_l \odot \theta_l) \mathcal{L}(g(A, X; \theta \odot m), y)$ for UGTs and the gradient norm of $\nabla \theta_L \mathcal{L}(g(A, X; \theta), y)$ for dense GCN where $l$ denotes the layer. Results are reported in Figure 14.

As we can observe, the gradient vanishing problem may exist for training deep dense GCN since the gradient norm for dense GCN is extremely small while UGTs does not have this problem. This problem might also be indicated by the training loss where the training loss for dense GCN does not
decrease while the training loss for UGTs decreases a lot. This might explain why UGTs performs well for deep GNNs.

![Figure 14: Gradient norm w.r.t each layer during training.](image)

**Figure 14:** Gradient norm w.r.t each layer during training. Experiments are conducted on Cora with GCN architecture containing 32 layers and width 448.

### B.7 More experiments for mitigating over-smoothing problem

We conduct experiments on Cora, Citeseer and Pubmed for GAT with deeper layers. The width is fixed to 448. The results of the other methods are obtained by running the code.

The results are reported in Table 6. It can be observed again that UGTs consistently outperforms all the baselines.

**Table 6:** Test accuracy (%) of different training techniques. The experiments are based on GAT models with 16, 32 layers, respectively. Width is set to 448.

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-Layers</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Trained Dense GAT</td>
<td>20.6</td>
<td>20.0</td>
<td>17.9</td>
</tr>
<tr>
<td>+Residual</td>
<td>19.9</td>
<td>17.7</td>
<td>41.6</td>
</tr>
<tr>
<td>+Jumping</td>
<td>39.7</td>
<td>29.1</td>
<td>57.3</td>
</tr>
<tr>
<td>+NodeNorm</td>
<td>70.9</td>
<td>17.1</td>
<td>72.2</td>
</tr>
<tr>
<td>+PairNorm</td>
<td>27.9</td>
<td>22.8</td>
<td>73.0</td>
</tr>
<tr>
<td>+DropNode</td>
<td>23.6</td>
<td>18.8</td>
<td>26.7</td>
</tr>
<tr>
<td>+DropEdge</td>
<td>24.8</td>
<td>19.4</td>
<td>19.3</td>
</tr>
<tr>
<td>UGTs-GAT</td>
<td>76.7 ± 1.1</td>
<td>74.9 ± 0.2</td>
<td>77.9 ± 0.5</td>
</tr>
</tbody>
</table>

### C Pseudocode

Pseudocode is showed in Algothrim 1.

---

1https://github.com/VITA-Group/Deep_GCN_Benchmarking.git
Algorithm 1 Untrained GNNs Tickets (UGTs)

**Input:** a GNN $g(A, X; \theta)$, initial mask $m = 1 \in \mathbb{R}^{[\theta]}$ with latent scores $S$, learning rate $\lambda$, hyperparameters for the gradual sparsification schedule $s_i, s_f, t_0,$ and $\Delta t$.

**Output:** $g(A, X; \theta \odot m), y$

Randomly initialize model weights $\theta$ and $S$.

for $t = 1$ to $T$ do

#Calculate the current sparsity level $s_t$ by Eq. 5.

$s_t \leftarrow s_f + (s_i - s_f)(1 - \frac{t - t_0}{\tau \Delta t})$

#Get the global threshold value at top $s_t$ by sorting $S$ in ascending order.

$S_{thres} \leftarrow Thresholding(S, s_t)$

#Generate the binary mask.

$m \leftarrow 0$ if $S < S_{thres}$ else 1

#Update $S$.

$S \leftarrow S - \lambda \nabla_S \mathcal{L}(g(A, X; \theta \odot m), y)$

end for

Return $g(A, X; \theta \odot m), y$