DOTIN: DROPPING OUT TASK-IRRELEVANT NODES FOR GNNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Scalability is an important consideration for deep graph neural networks. Inspired by the conventional pooling layers in CNNs, many recent graph learning approaches have introduced the pooling strategy to reduce the size of graphs for learning, such that the scalability and efficiency can be improved. However, these pooling-based methods are mainly tailored to a single graph-level task and pay more attention to local information, limiting their performance in multi-task settings which often require task-specific global information. In this paper, departure from these pooling-based efforts, we design a new approach called DOTIN (Dropping Out Task-Irrelevant Nodes) to reduce the size of graphs. Specifically, by introducing K learnable virtual nodes to represent the graph embeddings targeted to K different graph-level tasks, respectively, up to 90% raw nodes with low attentiveness with an attention model – a transformer in this paper, can be adaptively dropped without notable performance decreasing. Achieving almost the same accuracy, our method speeds up GAT about 50% on graph-level tasks including graph classification and graph edit distance (GED) with about 60% less memory, on D&D dataset.

1 INTRODUCTION

In recent years, there has been a surge of interest in developing graph neural networks (GNNs) to extract semantic features of graph-structured data, such as social network data (Xu et al., 2018; Kipf & Welling, 2016; Veličković et al., 2017) and graph-based representations of molecules (Dai et al., 2016). The success behind GNNs (against MLPs) is message passing between nodes by using (task-)specific prior knowledge of node adjacency (Zhang et al., 2020). However, the message passing also brings $\mathcal{O}(N^2)$ complexity (each node computes the weighted average embeddings of its neighbors). To reduce the complexity and enhance the model's scalability, for graph-level tasks, one plausible solution is narrowing down the graph size hierarchically, which can be fulfilled by graph pooling, as the counterpart to that in CNNs (LeCun et al., 2015).

Graph pooling methods (Ying et al., 2018; Cangea et al., 2018; Gao & Ji, 2019; Lei et al., 2019) usually stack GNN and pooling layers, which extract local (sub-graph) representations from several neighbor nodes. Previous pooling methods mainly differ in how to divide sub-graphs (Ying et al., 2018; Cangea et al., 2018) and assign pooling weights of nodes (Gao & Ji, 2019; Cangea et al., 2018). Then, the extracted sub-graph representations are fed into the next GNN layer to reduce the overall graph size. The whole architecture of the pooling-based GNN is akin to CNNs, which use convolution layers as filters and pooling layers to increase the receptive field. However, these pooling methods show defects in two aspects. **i) structure design**, pooling methods over-emphasize the local information (Dosovitskiy et al., 2020), but ignore global interaction which is useful for down-stream tasks (classification (Dosovitskiy et al., 2020)); **ii) learning paradigm**, pooling schemes are commonly used in GNNs which are task-agnostic, since they don't use the global representation to select nodes to drop, and correspondingly, they can **not** learn which nodes are *task-irrelevant* for specific tasks, as illustrated in Fig. 1 and in our later ablation studies.

In this paper, inspired by the recent transformer-based methods (Dosovitskiy et al., 2020; Liang et al., 2022), we propose DOTIN to tackle the above two problems. Specifically, DOTIN uses K virtual nodes to directly capture global information targeted to K different tasks (each virtual node learns one task-specific global information). Then, the virtual nodes compute the mean attentiveness of the K tasks, and adaptively select to drop task-irrelevant nodes over the whole graph. In this paper, task-irrelevant nodes are defined as those with lower attentiveness by the

softmax-based attention mechanism as will be shown in the technical part of the paper. Similar meanings for edges are also termed in (Zheng et al., 2020). The highlights of the paper are:

1) We show the phenomenon, i.e., in existing GNNs architecture, for different tasks, *task-irrelevant* nodes may be similar but not the same (illustrated in Fig. 1). The proposed DOTIN can find those task-irrelevant nodes and sub-graphs in both single-task and multi-task settings. As far as we know, DOTIN is the first for proposing using virtual nodes for multi-task learning.

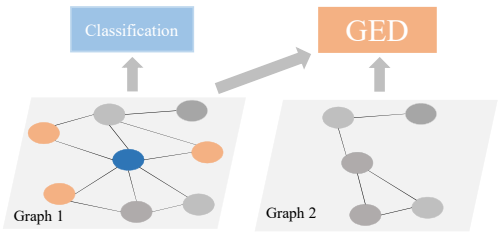


Figure 1: Motivation of multiple virtual nodes setting. For single-graph tasks like graph classification, we may emphasize the center node (blue) of the graph. However, for pair-graph tasks like graph matching or other comparison tasks, the three orange nodes can be more important.

2) To our best knowledge, this is a new paradigm (agnostic to the choice of GNNs architecture e.g. GAT (Veličković et al., 2017), GCN (Kipf & Welling, 2016)) to reduce the graph size by dropping nodes hierarchically, while previous methods almost adopt graph pooling. Besides, DOTIN only incurs $\mathcal{O}(N \log N)$ complexity by introducing and sorting K vectors without any other parameters. In contrast, previous methods usually require extra clustering (Ying et al., 2018), or GNN (Bianchi et al., 2020; Ying et al., 2018) and GCN (Lee et al., 2019) layers that will bring much more time and space complexity than sorting.

3) We conduct experiments on both single-task and multi-task settings on benchmarks. Results show that our methods outperform peer pooling methods. In particular, DOTIN drops 90% nodes without performance drop against the baseline and achieves about 50% training speed gain on D&D dataset.

2 Related Work

Here we briefly discuss the relevant works on cost-efficient GNN design and computing by edge/node drop and effective pooling. More comprehensive reviews are given in the appendix.

Graph backbone and aggregation functions. Various GNN backbones (Kipf & Welling, 2016; Veličković et al., 2017; Hamilton et al., 2017) have been devised to capture graph structural information. They mainly differ in the specific aggregation functions. As a comprehensive study, GraphSAGE (Hamilton et al., 2017) adopts four different aggregation methods, namely max, mean, GCN (Kipf & Welling, 2016), and LSTM (Hochreiter & Schmidhuber, 1997). Instead, Graph Attention Networks (Veličković et al., 2017) proposes attention-based methods (Vaswani et al., 2017), where the edge weights are learned by network adaptively. Graph Isomorphism Networks (GINs) (Xu et al., 2018) proves that GNN can satisfy the 1-Weisfeiler-Lehman (WL) condition only with sum pooling function as aggregation function. Recently, DeeperGCN (Li et al., 2020) proposes a trainable softmax and power-mean aggregation function that generalizes basic operators.

Graph pooling. The mainstream of graph pooling methods can be divided into global and hierarchical approaches (Mesquita et al., 2020). Global methods aggregate all nodes' representations either via simple flatten schemes, such as summation and average (Kipf & Welling, 2016; Veličković et al., 2017). While hierarchical approaches coarsen graph representations layer-by-layer. DiffPool (Ying et al., 2018) is the seminal work of hierarchical approach, which downsamples graphs by clustering nodes in input graphs, and computes the assignment matrix in the l-th layer $\mathcal{S}^{(l)}$ with learned clusters. Specifically, nodes in the l-th layer are assigned by:

$$\mathbf{S}^{(l)} = \texttt{Softmax}\left(\texttt{GNN}_l(\mathbf{A}^{(l)}, \mathbf{X}^{(l)})\right), \quad \mathbf{A}^{(l+1)} = \mathbf{S}^{(l)\top}\mathbf{A}^{(l)}\mathbf{S}^{(l)} \tag{1}$$

where $\mathbf{X}^{(l)}$ and $\mathbf{A}^{(l)}$ are the node features and adjacency matrix of the i-th layer. Hence, a clustering complexity will be introduced. gPool (Gao & Ji, 2019) designs a learnable vector to choose nodes to be retained. However, there's no regularization on the learnable vector, which may inappropriately delete *task-relevant* and important nodes. SAGPool (Lee et al., 2019) addresses this issue by using a graph convolution layer, followed by Sigmoid function to learn which nodes should be masked. Nevertheless, it introduces a GNN layer parameters and may ignore task-relevant information.

Edge/node drop. Edge drop is often adopted as a regularization technique in GNNs, especially for preventing from over-smoothing and for better generalization (Rong et al., 2019; Zheng et al.,

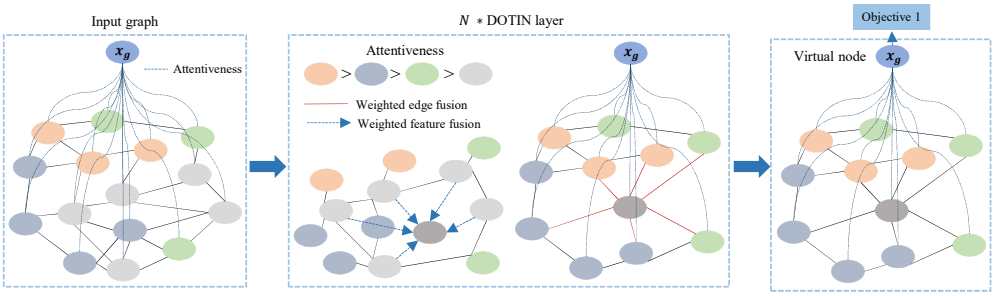


Figure 2: Framework of DOTIN for single task setting. The global virtual node is initially fully connected. Then five nodes in dash gray are fused in one global sub-global nodes. The new node (fused) reserves both structure (edges in red line) and statistic information (feature fusion).

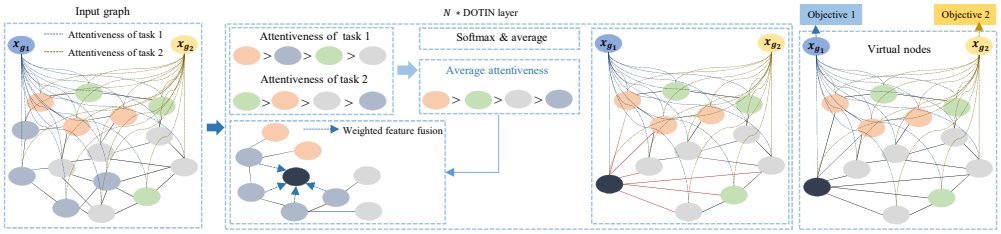


Figure 3: Framework of DOTIN for multiple-task setting (two tasks). Both global virtual nodes are initially fully-connected with Gaussian distribution. Then the mean attentiveness is used for replacing the attentiveness in each task. We also use one new node to represent the dropped sub-graph, which is similar to single task in Fig. 2. The overall objective is the sum of two single-task objectives.

2020). However, they can rarely help improve the scalability of the GNN model as the node size does not change. Accordingly, node drop is recently considered in DropGNN (Papp et al., 2021), which is in fact the only node dropping work so far we have identified. DropGNN proposes to drop nodes randomly several times and ensemble each predictive results. However, by the DropGNN learning paradigm, some important nodes may be dropped, which limits their performance and stability. Besides, multiple running and ensemble the prediction of each dropped graph also increase the complexity and training time. Inspired by their work, give one or multiple specific task(s), it is more attractive to develop adaptive mechanism to select the task-irrelevant nodes for dropping in one shot.

3 Methodology

3.1 Preliminaries

We first briefly review the propagation mechanisms as used in GNNs regardless the specific backbone choice e.g. GAT (Veličković et al., 2017) or GCNs (Kipf & Welling, 2016) and we will compare them in our experiments. Denote one input graph as $\mathcal{G} = \{\mathbf{X}^{(0)}, \mathbf{A}^{(0)}\}$, where $\mathbf{X}^{(0)} \in \mathbb{R}^{N\times F}$ and $\mathbf{A}^{(0)} \in \mathbb{R}^{N\times N}$ are input feature matrix and adjacency matrix, respectively. Graph convolution propagates graph by:

$$\mathbf{X}^{(l+1)} = \sigma\left((\mathbf{D}^{(l)})^{-1/2}\hat{\mathbf{A}}^{(l)}(\mathbf{D}^{(l)})^{-1/2}\mathbf{X}^{(l)}\theta^{(l)}\right), \quad \mathbf{A}^{(l+1)} = \mathbf{A}^{(l)}, \tag{2}$$

where $\mathbf{X}^{(l)}$ and $\mathbf{A}^{(l)}$ is the feature and adjacency matrix in the l-th layer, respectively. $\hat{\mathbf{A}}^{(l)} \in \mathbb{R}^{N\times N}$ is the adjacency matrix with self-loop. $\mathbf{D}^{(l)}$ is the degree matrix of $\hat{\mathbf{A}}^{(l)}$. $\theta^{(l)}$ is the learnable linear parameters of the l-th layer. Rather than use pre-defined edge weights, GAT tends to learn weights by itself, and the weights are computed by:

$$\mathbf{S}^{(l)} = \texttt{Softmax}\left((\mathbf{X}^{(l)}\mathbf{W}_1^{(l)}(\mathbf{X}^{(l)}\mathbf{W}_2^{(l)})^\top) \odot \hat{\mathbf{A}}\right) \tag{3}$$

where $\odot$ is the element-wise product, and $\mathbf{W}^{(l)}$ are the learnable weights of the l-th layer.

3.2 The Proposed DOTIN

The key idea behind DOTIN is adding virtual nodes. Then, the graph-level objectives (K objectives for K tasks, respectively) regularize the virtual nodes one by one, forcing them to learn task-relevant

information. Then, by calculating the average attentiveness of the virtual nodes and raw nodes, we know which nodes are task-irrelevant to these tasks.

Single-task virtual node. The input feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ is firstly passed by a linear projection layer $\mathbf{X}^{(0)} = \mathbf{X}^{(0)}\mathbf{W}$, where $\mathbf{W} \in \mathbb{R}^{F \times D}$ and D is the hidden dimension. Then a virtual node is initialized as a learnable vector, which is connected to all the other nodes in the graph. Then, the input feature matrix and matrix becomes $\mathbf{X}^{(0)} \in \mathbb{R}^{(N+1) \times D}$ and $\mathbf{A}^{(0)} \in \mathbb{R}^{(N+1) \times (N+1)}$, respectively, where the first raw feature of $\mathbf{X}^{(0)}$ is the virtual node embedding. Then, the propagation is in line with the applied backbones (e.g. GCNs, GATs, etc.). For the last layer, we directly add the objective regularization on the virtual nodes. For GATs-based backbones, the propagation of global virtual node and objective are as follows, respectively:

$$\mathbf{x}_g^{(l+1)} = \sum_i \left(\frac{\exp(\mathbf{x}_g^{(l)\top} \mathbf{x}_i^{(l)})}{\sum_j \exp(\mathbf{x}_g^{(l)\top} \mathbf{x}_j^{(l)})} \cdot \mathbf{x}_i^{(l)} \right), \quad \mathcal{J} = \mathcal{L}_{cls,reg}(f(\mathbf{x}_g), y), \tag{4}$$

where $\mathbf{x}_g$ is the embedding of virtual node (global embedding) and $\mathbf{x}_i$ is the i-th node of input graph. y is the graph-level task label, and f is the last linear layer to apply to different down-stream tasks.

Multi-task virtual nodes. For K different graph-level tasks, we initialize K learnable global virtual nodes with random distribution, where each virtual nodes are connected with the whole graph. The propagation rule is the same with single-task setting in hidden layers. Then, for the last layer, the propagation and objectives become to:

$$\mathbf{x}_{g_k}^{(l+1)} = \sum_i \left(\frac{\exp(\mathbf{x}_{g_k}^{(l)\top} \mathbf{x}_i^{(l)})}{\sum_j \exp(\mathbf{x}_{g_k}^{(l)\top} \mathbf{x}_j^{(l)})} \cdot \mathbf{x}_i^{(l)} \right), \quad \mathcal{J} = \frac{1}{K} \sum_i \mathcal{L}_{cls,reg}(f(\mathbf{x}_{g_i}), y_i), \tag{5}$$

where $\mathbf{x}_{g_i}$ and y_i are the global representation and ground truth for the i-th task.

Assumption 1 *Given one task-specific global representation $\mathbf{x}_g$ of a graph, if the nodes embedding $sim(\mathbf{x}_1, \mathbf{x}_g) > sim(\mathbf{x}_2, \mathbf{x}_g)$, we assume node $\mathbf{x}_1$ is more task-relevant than $\mathbf{x}_2$.*

Node dropping by attentiveness. Assume in the l-th layer, there are $N + K$ nodes in input graph (including K virtual nodes). Give a pre-defined drop ratio $0 < \alpha^{(l)} < 1$ for layer l, there are $M = \lceil N \cdot (1 - \alpha) + K \rceil + 1$ nodes remained after dropping the raw nodes, including the K virtual nodes and one sug-global node, which we will describe in detail below. Inspired by EViT (Liang et al., 2022), we calculate the attentiveness of virtual nodes as well as the remaining raw nodes by:

$$s_i = \sum_k (\mathbf{x}_{g_k} \mathbf{W}_1)^\top (\mathbf{x}_i \mathbf{W}_2), \quad \mathbf{s} = \texttt{Softmax}(\mathbf{s}/\tau), \quad 0 < i \leq N \tag{6}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are two learnable parameters in GATs (Veličković et al., 2017), τ is set $D^{1/2}$ by default. After obtaining the attentiveness of all the remaining raw nodes, by Assumption 1, we sort the attentiveness and find the index of the smallest $\lfloor N \cdot \alpha \rfloor$ number of nodes, which are called *task-irrelevant* nodes in this paper. Then we introduce a new node to represent the sub-graph composed of these task-irrelevant nodes. The feature of the new node is obtained by mixing the task-irrelevant nodes with different weights (attentiveness), i.e.,

$$\mathbf{x}'_{\lceil N \cdot (1-\alpha) \rceil + 1} = \sum_i^{\lfloor N \cdot \alpha \rfloor} \lambda_i \cdot \mathbf{x}_i, \quad \lambda = \texttt{Softmax}([s[1], \cdots, s[\lfloor N \cdot \alpha \rfloor]]) \tag{7}$$

To further preserve the structural information of the sub-graph, the new node connects all nodes connected with the task-irrelevant sub-graph and the edge weights are modified by:

$$w'_{i, \lceil N \cdot (1-\alpha) \rceil + 1} = \sum_j^{\lfloor N \cdot \alpha \rfloor} w_{i,j}, \quad \mathbf{w}_{i,:} = \texttt{Softmax}(\mathbf{w}_{i,:}) \tag{8}$$

where $\mathbf{w}_{i,:}$ is the vector composed of edge weights of node i and the other nodes. This design is motivated by: 1) if two or more nodes in the task-irrelevant sub-graph connect nodes in the reserved graph, we think the node is more related to the task-irrelevant sub-graph, and the weight of new node with the node in reserve graph is the summation followed by a softmax regularization, which is more comprehensible than average (ignore the connection degree) and counting (ignore edge weights). Finally, we stack the propagation layers and node dropping stage one by one.

Backbone choose. Here, we briefly analyze why we prefer to choose GAT but not GCNs (or other methods with pre-defined edge weights, e.g., GIN (Xu et al., 2018) and GraphSAGE (Hamilton et al.,

2017)) as the backbone. On one hand, the attentiveness in Eq. 6, requires two linear layers and a dot product operation, which are also components in GAT layer. Nevertheless, GCN layer doesn't include these operations, which makes DOTIN less efficient in GCN. On the other hand, GCN-based backbones are also unsuitable for multi-task settings, which we will explain in detail below. The forward propagation rule is given in Eq. 2, where $\mathbf{A}$ is updated in each layer. Consider K tasks with α drop ratio in a graph including N nodes, the output of DOTIN contains $K + 1 + \lfloor N \cdot (1 - \alpha) \rfloor$ nodes. Then, for the next layer, since each virtual node is initially fully-connected, their embedding is modified by:

$$\mathbf{x}_{g_k} = \frac{1}{2 + \lfloor N \cdot (1 - \alpha) \rfloor} \left(\sum_i \mathbf{x}_i + \mathbf{x}_{g_k} \right) \quad for \;\; 1 \leq k \leq K \tag{9}$$

We can observe the embeddings of virtual nodes are equivalent, which is contradictory to our initial desire, i.e., applying different virtual nodes to different tasks.

3.3 Theoretical Analysis

DOTIN does not directly drop the task-irrelevant sub-graph, instead it constructs a new node to reserve the structural information. Here, we theoretically analyze the motivation of our design.

Theorem 1 ***Attentive pooling for improving task-relevant node embedding.*** *Define the distance of two mode feature vectors by $l_2(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2$. Given a graph learned with a given graph-level task $\mathcal{T}$ e.g. graph classification with node embeddings $\{\mathbf{x}_i\}_{i=1}^N$ of its sub-graph and the global graph-level embedding $\mathbf{x}_g$ learned from the given task, where N is the number of nodes in the graph. Then, the nodes $\mathbf{x}_{new}$ produced by attentive pooling has closer distance to $\mathbf{x}_g$ than the average distance of the sub-graph before pooling, i.e., $l_2(\mathbf{x}_g, \mathbf{x}_{new}) < \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i l_2(\mathbf{x}_g, \mathbf{x}_i)$.*

Proof *To prove the theorem, we first prove the average pooling can reduce the distance, and in the second step, we prove the weighted by attentiveness pooling will construct a new node with larger attentiveness. By the definition, we have:*

$$\begin{aligned}
\frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \|\mathbf{x}_g \mathbf{W}_1 - \mathbf{x}_i \mathbf{W}_2\|_2^2 &= \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \left(\|\mathbf{x}_g \mathbf{W}_1\|_2^2 + \|\mathbf{x}_i \mathbf{W}_2\|_2^2 - 2 \cdot (\mathbf{x}_g \mathbf{W}_1)^\top (\mathbf{x}_i \mathbf{W}_2) \right) \\
&= \left\| \mathbf{x}_g \mathbf{W}_1 - \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \mathbf{x}_i \mathbf{W}_2 \right\|_2^2 - \left\| \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \mathbf{x}_i \mathbf{W}_2 \right\|_2^2 + \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \|\mathbf{x}_i \mathbf{W}_2\|_2^2 \\
&= \left\| \mathbf{x}_g \mathbf{W}_1 - \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \mathbf{x}_i \mathbf{W}_2 \right\|_2^2 + \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \left\| \frac{1}{\lceil N \cdot \alpha \rceil} \sum_j \mathbf{x}_j \mathbf{W}_2 - \mathbf{x}_i \mathbf{W}_2 \right\|_2^2
\end{aligned} \tag{10}$$

where the first term equals to $l_2(\mathbf{x}_g \mathbf{W}_1, avg(\mathbf{x}_i) \mathbf{W}_2)$ and the second term equals to $avg(l_2(\mathbf{x}_i \mathbf{W}_2, avg(\mathbf{x}_j) \mathbf{W}_2))$, which are thus both non-negative. For shallow GNNs, the node features are usually different, leading the second term R.H.S larger than 0. Then, we derive $\frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \|\mathbf{x}_g \mathbf{W}_1 - \mathbf{x}_i \mathbf{W}_2\|_2^2 \geq \|\mathbf{x}_g \mathbf{W}_1 - \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \mathbf{x}_i \mathbf{W}_2\|_2^2$, i.e., the averagely constructed node has closer distance than the average distance of the task-irrelevant sub-graph. Then, we begin proving the weighted pooling by attentiveness will bring higher attentiveness. As defined in Eq. 6, The attentiveness is calculated by $s_i = (\mathbf{x}_g \mathbf{W}_1)^\top (\mathbf{x}_i \mathbf{W}_2)$ followed by a softmax. For average sub-graph pooling, we have $s = (\mathbf{x}_g \mathbf{W}_1)^\top (\frac{1}{\lceil N \cdot \alpha \rceil} \sum_i \mathbf{x}_i \mathbf{W}_2) = \frac{1}{\lceil N \cdot \alpha \rceil} \sum_i (\mathbf{x}_g \mathbf{W}_1)^\top (\mathbf{x}_i \mathbf{W}_2)$, while for weighted sub-graph pooling, we have $s' = (\mathbf{x}_g \mathbf{W}_1)^\top (\mathbf{x}_{new} \mathbf{W}_2) = \sum_i (\mathbf{x}_g \mathbf{W}_1)^\top (\lambda_i \mathbf{x}_i \mathbf{W}_2)$. For $(\mathbf{x}_g \mathbf{W}_1)^\top (\mathbf{x}_i \mathbf{W}_2) > (\mathbf{x}_g \mathbf{W}_1)^\top (\mathbf{x}_j \mathbf{W}_2)$, we must have $\lambda_i > \lambda_j$ (attentiveness definition), while by the regularization in Eq. 7, we have $\sum_i \lambda_i = 1$. Thus, we derive $s' > s$ and complete the proof.

Theorem 1 clarifies the motivation of task-irrelevant pooling methods, i.e., the newly constructed node has higher attentiveness and closer distance to the global virtual nodes.

3.4 Methodology Discussion

We compare DOTIN with previous pooling-based methods in Table 1. Here, we further clarify the connection and differences with similar methods gPool and SAGPool. **Connection:** All the three methods use attention methods to calculate the importance and adaptively choose task-irrelevant nodes to be dropped. **Differences:** The main difference between DOTIN and them is that DOTIN can recognize **task-irrelevant** nodes and the *task-irrelevant* sub-graph pooling also makes DOTIN achieve higher accuracy. Moreover, DOTIN only introduces K extra vectors while the previous two methods (gPool and SAGPool) have to learn extra GNN / GCN layers, incurring additional overhead.

	Time	Space	Task-relevant	Reduce method (Grattarola et al., 2021)	Extra parameters
Set2Set (Vinyals et al., 2015)	LSTM	LSTM	No	$\mathbf{X}' = \mathbf{S} \cdot GNN(\mathbf{A}, \mathbf{X})$	LSTM layer
DiffPool (Ying et al., 2018)	GNN + CLuster	GNN	No	$\mathbf{X}' = \mathbf{S} \cdot GNN(\mathbf{A}, \mathbf{X})$	GNN layer
MinCut (Bianchi et al., 2020)	Cluster + GNN + MLP	GNN + MLP	No	$\mathbf{X}' = \mathbf{S}^\top \mathbf{X}$	MLP + GNN layer
LaPool (Noutahi et al., 2019)	Cluster + Linear	Linear	No	$\mathbf{X}' = \mathbf{S}^\top \mathbf{X}$	Linear layer
gPool (Gao & Ji, 2019)	$\mathcal{O}(N \log N)$	$\mathcal{O}(N)$	No	$\mathbf{X}' = (\mathbf{X} \odot \sigma(\mathbf{y}))$	GNN layer
SAGPool (Lee et al., 2019)	GCN + $\mathcal{O}(N \log N)$	GCN	No	$\mathbf{X}' = (\mathbf{X} \odot \sigma(\mathbf{y}))$	GCN layer
DOTIN (Ours)	$\mathcal{O}(N \log N)$	$\mathcal{O}(1)$	Yes	$\mathbf{X}' = (\mathbf{X} \odot \sigma(\mathbf{y}))$	$\mathcal{O}(1)$

Table 1: Methodology comparison. The time and space complexity are relevant to clustering iteration, GNN, MLP layers and their dimensions. gPool (Gao & Ji, 2019) and DOTIN (ours) only use an extra rank time complexity. Space $\mathcal{O}(1)$ means we only use K global vectors ($K = 1$ when single-task setting). Task-relevant means whether the method is task-specific or not.

Dataset	# Graphs	# Classes	Avg. Nodes per Graph	Avg. Edges per Graph	# Training	# Test
D&D (Dobson & Doig, 2003)	1178	2	284.32	715.66	1060	118
PROTEINS (Borgwardt et al., 2005)	1113	2	39.06	72.82	1001	112
NCI1 (Wale et al., 2008)	4110	2	29.87	32.3	3699	411
NCI109 (Wale et al., 2008)	4127	2	29.68	32.13	3714	413
FRANKENSTEIN (Orsini et al., 2015)	4337	2	16.9	17.88	3903	434

Table 2: Statistic information of the used datasets.

4 EXPERIMENTS

We evaluate our method on graph classification and graph edit distance in single-task and multi-task settings, respectively. We conduct ablation studies on memory and time consumption. Experiments are mainly in line with the protocol of gPool and the mean and standard deviation are reported by 10-fold cross validation, except for the memory and time tests which can be estimated by one trial. We use GAT as backbone except for ablation in Tab. 6 as analyzed in Section 3.2 and all the experiments are conducted on one single GTX 2080 GPU. The dataset details is given in Appendix A

4.1 EXPERIMENTS SETUP

Graph classification task. For all the five graph-classification datasets, we set the learning rate as 1e-3 with batch size 8 and use Adam optimizer (Kingma & Ba, 2014) with weight decay 8e-4.

Graph edit distance (GED) task. i) setup. The GED between graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is defined as the minimum number of edit operations needed to transform $\mathcal{G}_1$ to $\mathcal{G}_2$. Typically the edit operations include add/remove/substitute nodes and edges. Computing GED is known NP-hard in general, therefore approximations are used. There are also attempts by deep graph model and we adopt the same setting. In detail, triplet pairs are constructed by editing graph (substitute and remove) edges, which is an unsupervised model. Specifically, they substitute k_p edges from graph $\mathcal{G}_1$ to generate $\mathcal{G}_{1p}$, then substitute k_n edges to generate $\mathcal{G}_{1n}$. By setting $k_p < k_n$, the GED between $(\mathcal{G}_1, \mathcal{G}_{1p})$ is regarded as shorter than $(\mathcal{G}_1, \mathcal{G}_{1n})$. But actually, the GED between $(\mathcal{G}_1, \mathcal{G}_{1p})$ can be smaller than $(\mathcal{G}_1, \mathcal{G}_{1n})$ due to symmetry and isomorphism. However, the probability of such cases is typically low and decreases rapidly with increasing graph sizes. **ii) evaluation metrics.** Followed (Li et al., 2019), our trained backbone is evaluated by two metrics: 1) pair AUC - the area under the ROC curve for classifying pairs of graphs as similar or not and 2) triplet accuracy - the accuracy of correctly assigning higher similarity to the positive pair than the negative pair, in a triplet.

Compared baselines. Since our method aims to reduce the graph size hierarchically, we mainly compare with recent hierarchical graph pooling methods: DiffPool (Ying et al., 2018), gPool (Gao & Ji, 2019) and SAGPool (Lee et al., 2019) as they can readily allow for node dropping in their methods. Note we do not compare with other recent methods e.g. GIN (Xu et al., 2018), GraphSAGE (Hamilton et al., 2017), as the methodology and motivation are different from our node dropping-based scalability-purposed methodology. We also compare with our degenerated baseline, namely using the same backbone as DOTIN (i.e. GAT) but without node drop. The metrics include both accuracy and training time. For comprehensive comparison, on graph classification, we further compare global pooling methods Set2Set (Vinyals et al., 2015), SortPool (Zhang et al., 2018), SAGPool (Lee et al., 2019), for which node dropping cannot be (easily) fulfilled. Note that we don't directly compare DOTIN with DropGNN (Papp et al., 2021), and the reason is that DropGNN is an ensemble model and its variant single model can be thought one of our baseline (random drop), which is in-depthly compared in ablation studies (see Fig. 4(a)).

	Models	D&D	PROTEINS	NCI1	NCI109	FRANKENSETEIN
Global	Set2Set (Vinyals et al., 2015)	71.27 ± 0.84	66.06 ± 1.66	68.55 ± 1.92	69.78 ± 1.16	61.92 ± 0.73
	SortPool (Zhang et al., 2018)	72.53 ± 1.19	66.72 ± 3.56	73.82 ± 0.96	74.02 ± 1.18	60.61 ± 0.77
	SAGPool (Lee et al., 2019)	**76.19 ± 0.94**	**70.04 ± 1.47**	**74.18 ± 1.20**	**74.06 ± 0.78**	**62.57 ± 0.60**
Hierarchical	DiffPool (Ying et al., 2018)	66.95 ± 2.41.	68.20 ± 2.02	62.32 ± 1.90	61.98 ± 1.98	60.60 ± 1.62
	gPool (Gao & Ji, 2019)	75.01 ± 0.86	71.10 ± 0.90	67.02 ± 2.25	66.12 ± 1.60	61.46 ± 0.84
	SAGPool (Lee et al., 2019)	76.45 ± 0.97	71.86 ± 0.97	67.45 ± 1.11	**67.86 ± 1.41**	61.73 ± 0.76
	DOTIN (w/o pooling)	77.41 ± 0.72	73.50 ± 0.5	68.12 ± 1.07	67.27 ± 1.12	62.41 ± 1.12
	DOTIN (w/ pooling)	**78.25 ± 0.61**	**74.63 ± 0.37**	**69.39 ± 1.02**	67.62 ± 1.04	**63.01 ± 0.92**

Table 3: Graph classification accuracy in 10 folds on benchmarks.

Model	D&D		PROTEINS		NCI1	
	ACC	AUC	ACC	AUC	ACC	AUC
DiffPool (Ying et al., 2018)	85.19 ± 0.52	72.29 ± 1.64	71.12 ± 1.08	54.44 ± 2.99	80.68 ± 1.71	59.98 ± 3.38
gPool (Gao & Ji, 2019)	88.26 ± 0.61	72.89 ± 1.66	72.17 ± 1.02	54.91 ± 2.27	85.54 ± 1.29	62.29 ± 2.98
SAGPool (Lee et al., 2019)	90.17 ± 0.44	73.01 ± 1.29	73.31 ± 1.14	59.18 ± 2.46	85.92 ± 1.19	64..48 ± 2.70
DOTIN (w/o pooling)	90.67 ± 0.31	73.09 ± 1.22	75.89 ± 0.69	58.81 ± 2.19	87.34 ± 0.84	65.45 ± 2.61
DOTIN (w/ pooling)	**91.88 ± 0.24**	**74.27 ± 1.13**	**76.76 ± 0.57**	**59.91 ± 2.01**	**88.38 ± 0.73**	**66.83 ± 2.28**

Table 4: Accuracy (ACC) and AUC score in 10 folds of solving graph edit distance on benchmarks.

Model	D&D			PROTEINS			NCI1		
	CLS	GED ACC	GED AUC	CLS	GED ACC	GED AUC	CLS	GED ACC	GED AUC
DiffPool (Ying et al., 2018)	64.67 ± 2.92	81.17 ± 0.77	68.23 ± 1.99	65.20 ± 2.18	69.47 ± 1.42	52.41 ± 2.06	59.92 ± 1.97	74.15 ± 1.86	54.16 ± 3.21
gPool (Gao & Ji, 2019)	72.06 ± 0.99	89.38 ± 0.68	69.38 ± 1.79	69.92 ± 0.94	70.25 ± 1.16	52.77 ± 2.09	64.28 ± 2.49	82.47 ± 1.67	57.37 ± 3.06
SAGPool (Lee et al., 2019)	75.53 ± 1.04	88.31 ± 0.57	71.39 ± 1.44	70.61 ± 0.93	71.16 ± 1.37	57.18 ± 2.16	68.19 ± 1.32	81.92 ± 2.24	59.94 ± 2.99
DOTIN (w/o pooling)	77.36 ± 0.78	90.49 ± 0.38	73.01 ± 1.21	73.41 ± 0.62	75.24 ± 0.57	58.44 ± 2.21	67.99 ± 1.14	86.99 ± 0.92	64.56 ± 2.73
DOTIN (w/ pooling)	**78.14 ± 0.72**	**91.49 ± 0.31**	**74.16 ± 1.12**	**74.43 ± 0.49**	**76.16 ± 0.42**	**59.16 ± 2.03**	**68.92 ± 1.09**	**87.97 ± 0.89**	**66.29 ± 2.24**

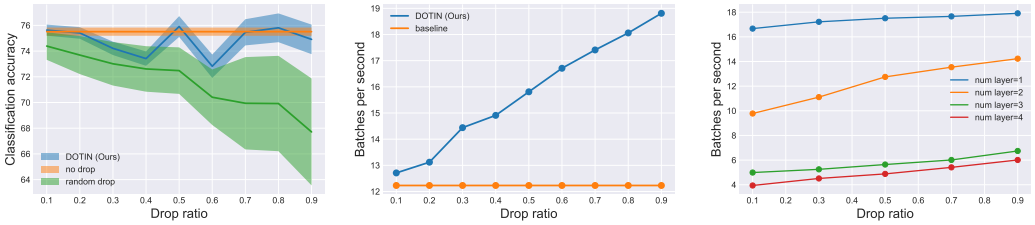
Table 5: Classification accuracy and GED AUC/accuracy under multi-task setting.

Single-task setting We first evaluate our method on graph classification (in Table 3) and graph edit distance (in Table 4) respectively. **For graph classification**, DOTIN outperforms on most of the datasets among the compared hierarchical methods. Compared with the most similar method gPool (Gao & Ji, 2019), DOTIN outperforms it on all the five datasets. Although global pooling methods (SAGPool and SortPool) get higher accuracy on NCI dataset, they only perform pooling after the final GCN layer, and the running time and complexity won't decrease but increase compared with the baseline (w/o pooling). For graph edit distance task, we report the mean accuracy and AUC scores of 10 folds in Table 4, which is similar to graph classification. **For GED task**, DOTIN outperforms gPool (Gao & Ji, 2019) with a large range, and we guess the reason is that DOTIN directly adds the regularization on the virtual node. Then, the virtual node is used for node selection, where it tends to select GED-task-irrelevant nodes to drop. In contrast, gPool (Gao & Ji, 2019) does not add task-specific regularization on the designed vector, which may wrongly drop GED-task-important nodes. Besides, for both two tasks, DOTIN (w/ pooling) achieves higher accuracy and is more stable than w/o pooling, which is because the task-irrelevant pooling method reverses as much information as possible, while DOTIN (w/o pooling) simply deletes the sub-graph, losing some useful information.

Multi-task setting We combine classification and GED, summing up their two objectives with equal weights (see also Fig. 3) for graph learning. For gPool and SAGPool, they only extract one graph representation, so we directly add the two objective regularizations on the embedding. For DOTIN, we use two virtual nodes (for two tasks) to extract the respective task-relevant graph-level information. The drop ratio is set to $[0.1, 0.2, \cdots, 0.9]$. We randomly split each dataset into 10 folds and we set batch size 16 with 5 epochs in each fold. **Results.** Table 5 reports the mean accuracy of 10 folds, which shows DOTIN performs best in multi-task setting. For gPool and SAGPool, both classification and GED performances drop notably. Specifically, for D&D dataset, accuracy of DOTIN on multi-task settings and single-task settings only drop 0.09% and 0.39% accuracy on classification and GED, respectively. For other methods, they generally drop $2 \sim 5\%$ accuracy. We conjecture DOTIN's performance advantage is because the designed multiple virtual nodes extract and decouple different task-relevant global information to handle the corresponding task.

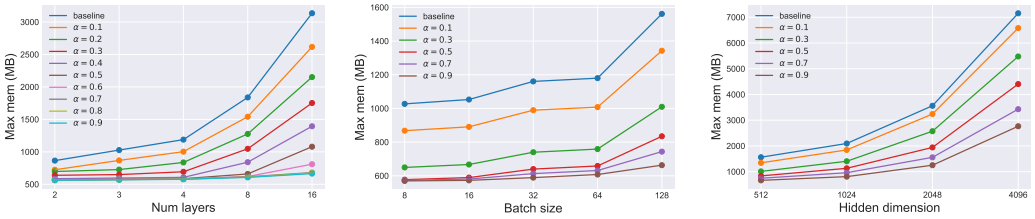
4.2 Ablation Study

Drop ratio effect. We conduct ablation studies on the drop ratio α. We construct three layers in the backbone and change the drop ratio from 0.1 to 0.9. We fix the hidden dimension as 512 and set the linear layers' connection dropout rate (Srivastava et al., 2014) 0.2 for inference (different from the meaning of the proposed node drop rate). We compare with baseline (w/o drop) and random drop. Fig. 4(a) and Fig. 4(b) show the classification accuracy and training speed (batch per second) of DOTIN with different drop ratios. Note that for random drop, the performance variance increases notably with higher drop ratios, but DOTIN is more stable even 90% of the nodes drop. For low drop ratio, random drop plays a role of regularization, which seems without much accuracy drop.



(a) CLS accuracy v.s. drop ratio (b) Speed v.s. drop ratio (c) Speed v.s. Num. of layers

Figure 4: Ablation on performance and batch training speed. Details can be seen in experiment setup.



(a) Memory v.s. $num\ layers$ (b) Memory v.s. batch size (c) Memory v.s. $hidden\ dim$

Figure 5: Ablation on memory. All the three experiments are using GAT backbone.

We also observe DOTIN occasionally outperforms baseline (w/o drop) sometimes, and we think this phenomenon may cause by node redundancy in original graphs, i.e., some task-irrelevant nodes influence the predictive accuracy. Fig. 4(b) gives the training time (batches per second) with different drop ratios. The baseline (w/o drop) passes average 12.23 batches per second for a whole training epoch. After dropping 90% nodes, DOTIN passes about 18.81 batches per second, which speeds up about 53.8% with little accuracy drop (74.91% v.s. 75.51%).

Number of layers. To explore the speed rate of DOTIN with a different number of layers, we conduct experiments on single GED tasks on D&D dataset. We fix the hidden dimension as 64 (enable to increase the number of layers) with ELU activation function (Clevert et al., 2015). We set the Num. of layers in [1, 3, 5, 7] and drop ratio as in [0.1, 0.3, 0.5, 0.7, 0.9]. We illustrate each two combination $(num\ layers, drop\ ratio)$ in Fig. 4(c). It's not surprising for one layer, the efficiency reduction is not significant, and this is because we drop the nodes after the first DOTIN layer. Then, the remaining nodes only pass through one linear layer. The overall FLOPs gap is only $\alpha \times D \times D_1$, where D and D_1 are input and output feature dimensions of the linear layer, respectively. For $num\ layer > 1$, we can find DOTIN with $\alpha = 0.9$ can speed up about 50% than $\alpha = 0.1$.

Backbone. DOTIN is agnostic to the backbone, and we integrate DOTIN into GCN (Kipf & Welling, 2016) as its backbone. Different from vallina GCNs, DOTIN in GCN calculates the normalized adjacency matrix in each layer, since in each layer, we drop $\lceil N \times \alpha \rceil$ nodes, and correspondingly, the adjacency matrix will be modified. In detail, we set $num\ layers = 2$ and hidden dimension as 256 for both GCN-based and GAT-based backbones. We find DOTIN in GCN-based backbone is more sensitive to drop ratio than GAT-based backbone, and we guess that's because GAT-based backbone can learn adjacency matrix by itself, which allows the classification virtual node to give higher edge weights to relevant nodes. While for GCN, the virtual node is initially fully-connected, i.e., in the message passing stage, the virtual node will take all the nodes in graph equally (all nodes contribute equally). Nevertheless, we can also observe that GCN leads to lower variance than GATs, which may be caused by the overfitting (GAT has two more linear parameters to learn in each layer, see Tab. 6).

Memory. We conduct a group of experiments by increasing hidden dimension, batch size, and $num\ layer$ with drop ratio α. We first analyze the effect of $num\ layers$, we fix batch size as 8 and hidden dimension as 512. Then, we switch the $num\ layers$ from [2, 3, 4, 8, 16] and the results are given in Fig. 5(a). We find that with 16 layers, baseline (w/o drop) spends maximal 3135.55MB memory, while for $\alpha = 0.9$, DOTIN only spends maximal 664.47MB memory, which is about 22% of baseline. However, as illustrated in Fig. 4(a), even DOTIN drops 90% nodes, the performance almost doesn't decrease. For Fig. 5(b), we fix the hidden dimension as 512 and $num\ layers$ as 3. Then, we change the batch size from 8 to 128 and drop ratios α from 0.1 to 0.9. For Fig. 5(c), we fix batch size as 128 and $num\ layers$ as 3, and increase the hidden dimension from 512 to 4096. For

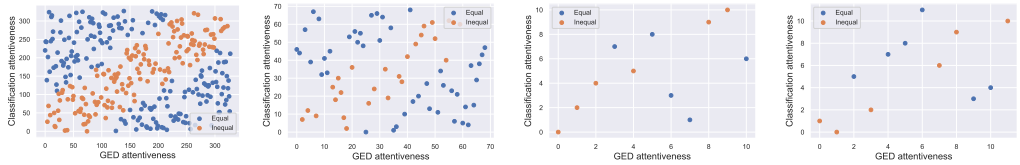


Figure 6: Scatter plot for importance ranking on D&D (left two examples) and PROTEINS (right two examples). The x axis is the ranked importance for GED and y for classification. Points close to diagonal (in orange) mean similar importance to two tasks (for orange, the attentive score ratio of the two tasks is at least 0.25), while those in blue indicate relevance to one task but less to the other.

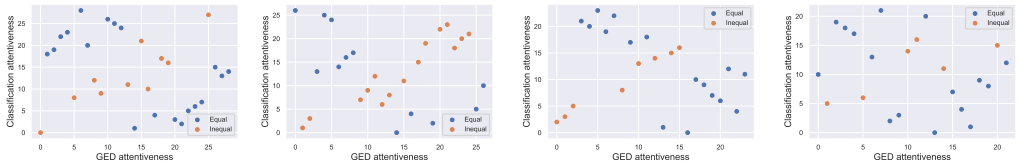


Figure 7: Importance ranking on NCI1 (left two examples) and NCI109 (right two examples) datasets.

Backbone	D&D	PROTEINS	NCI1	NCI109	FRANKENSETEIN
GCN ($\alpha = 0.1$)	73.69 ± 0.32	73.29 ± 0.37	66.18 ± 0.89	65.16 ± 1.14	60.21 ± 1.01
GCN ($\alpha = 0.5$)	72.81 ± 0.48	72.51 ± 0.64	64.93 ± 1.03	63.39 ± 1.51	58.19 ± 1.27
GCN ($\alpha = 0.9$)	71.17 ± 0.66	71.14 ± 0.71	63.37 ± 1.24	61.74 ± 1.79	54.49 ± 1.38
GAT ($\alpha = 0.1$)	75.63 ± 0.43	**73.41 ± 0.48**	**67.91 ± 1.09**	**67.11 ± 1.28**	**62.18 ± 1.17**
GAT ($\alpha = 0.5$)	**76.41 ± 0.69**	73.28 ± 0.61	67.58 ± 1.26	66.88 ± 1.67	61.26 ± 1.48
GAT ($\alpha = 0.9$)	75.99 ± 0.82	72.17 ± 0.79	66.98 ± 1.63	66.15 ± 1.99	60.99 ± 1.61

Table 6: Node classification accuracy of DOTIN's variants with different backbones and drop ratios.

$hidden\ dim = 512$, DOTIN with drop ratio 0.9 only spends 42% memory of baseline (w/o drop) and for $hidden\ dim = 4096$, DOTIN with 90% drop ratio reduces 4390MB memory over baseline.

Attentiveness for different tasks. As illustrated in Fig. 1, node importance can be different for different tasks. We visualize the attentiveness of classification and GED tasks from Fig. 6(a) to Fig. 7(d). For each dataset, we randomly select two graphs in test set for visualization. For training, we set hidden dimension as 256 and batch size as 8. For better comparison, we normalize the vectors before calculating the attention matrix. Then, we choose the first two rows (classification and GED virtual nodes) and sort by values. Then, we visualize the ranked number of the attentiveness. For each subplot, x axis is the GED importance rank and y axis classification importance rank. We can easily observe, the attentiveness for the two nodes with others are completely in different distribution, i.e., for different tasks, the *task-important* nodes are different (lots of points are closed to x or y axis). This phenomenon also explains DOTIN outperforms gPool (Gao & Ji, 2019), since gPool may drop *task-relevant* nodes, while DOTIN balances multiple tasks and drops nodes with averagely lower attentiveness. One of another interesting phenomenon is some nodes have the similar attentiveness to virtual nodes, we guess that is because of the *over smoothing* (Yang et al., 2020), which is a classical problems in deep GNNs.

5 Conclusion and Outlook

We have proposed DOTIN, which aims to enhance the efficiency and scalability of existing GNNs. Our method directly regularizes the introduced K virtual nodes with learnable vectors, corresponding to K tasks for learning, which helps drop *task-irrelevant* nodes. We apply DOTIN in GATs, with the extra cost of only $\mathcal{O}(1)$ parameters for learning. Experiments on graph classification and graph edit distance (GED) datasets show it achieves state-of-the-art results. Furthermore, DOTIN shows more advantages w.r.t cost-efficiency in multi-task settings (joint learning of graph classification and GED).

For future work, we aim to extend our experiments to more graph tasks beyond the current two-task setting, by exploring more graph-level tasks which in fact is so far seldom explored in GNNs literature (most graph-level learning works are focused on the two tasks: GED and classification as studied in this paper). This effort may also help better explore DOTIN's ability in multi-task learning, as well as its generalization ability to unseen tasks or new datasets, by considering node dropping as a certain way of improving generalization, beyond cost-efficiency and scalability as focused by this paper.

REFERENCES

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *ICML*, 2020.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1): i47–i56, 2005.

Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016.

Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 2003.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *ICML*, 2019.

Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *arXiv preprint arXiv:2110.05292*, 2021.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *NeurIPS*, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Katsuhiko Ishiguro, Shin-ichi Maeda, and Masanori Koyama. Graph warp module: an auxiliary module for boosting the power of graph neural networks in molecular graph analysis. *arXiv preprint arXiv:1902.01020*, 2019.

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

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *ICML*, 2019.

Huan Lei, Naveed Akhtar, and Ajmal Mian. Octree guided cnn with spherical kernels for 3d point clouds. In *CVPR*, 2019.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *ICML*, 2019.

Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations. *arXiv preprint arXiv:2202.07800*, 2022.

Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *NeurIPS*, 2020.

Emmanuel Noutahi, Dominique Beaini, Julien Horwood, Sébastien Giguère, and Prudencio Tossou. Towards interpretable sparse graph representation learning with laplacian pooling. *arXiv preprint arXiv:1905.11577*, 2019.

Francesco Orsini, Paolo Frasconi, and Luc De Raedt. Graph invariant kernels. In *IJCAI*, 2015.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: random dropouts increase the expressiveness of graph neural networks. *NeurIPS*, 2021.

Trang Pham, Truyen Tran, Hoa Dam, and Svetha Venkatesh. Graph classification via deep learning with virtual nodes. *arXiv preprint arXiv:1708.04357*, 2017.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2019.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 12(9), 2011.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. Revisiting over-smoothing in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *NeurIPS*, 31, 2018.

Li Zhang, Dan Xu, Anurag Arnab, and Philip HS Torr. Dynamic graph message passing networks. In *CVPR*, 2020.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. Robust graph representation learning via neural sparsification. In *ICML*, 2020.

A MORE DISCUSSION AND DATASET DETAILS

Readout function. The readout functions are widely designed by statistics e.g. min/max/sum/average of nodes to represent the graphs (Veličković et al., 2017; Kipf & Welling, 2016). On the basis of global pooling schemes, SortPooling (Zhang et al., 2018) chooses the top-k values from the sorted list of the node features to construct outputs. Instead, in this paper we propose to adopt a virtual node to help explicitly select the nodes to drop out. Interestingly, the term of *virtual node* is also called and used in VCN (Pham et al., 2017). In that work, the virtual node together with the real nodes are fed into an RNN for extracting the embedding of the whole graph[1]. The work (Ishiguro et al., 2019) also uses one virtual node with complex RNN-based models to extract global information, which is similar to VCN (Pham et al., 2017).

Datasets. D&D (Dobson & Doig, 2003; Shervashidze et al., 2011) contains graphs of protein structures. A node represents an amino acid and edges are constructed if the distance of two nodes is less than 6 $\mathring{A}$ (a unit of length in protein – see (Dobson & Doig, 2003)). A label denotes whether a protein is an enzyme or a non-enzyme. PROTEINS (Dobson & Doig, 2003; Borgwardt et al., 2005) also contains proteins, where nodes are secondary structure elements. If nodes have edges, the nodes are in an amino acid sequence or in a close 3D space. NCI (Wale et al., 2008) is a biological dataset used for anticancer activity classification. In the dataset, each graph represents a chemical compound, with nodes and edges representing atoms and chemical bonds, respectively. NCI1 and NCI109 are commonly used for graph classification (Veličković et al., 2017; Lee et al., 2019). FRANKENSTEIN (Orsini et al., 2015) is a set of molecular graphs with node features containing continuous values. The label denotes whether a molecule is a mutagen or not.

B LIMITATION AND POTENTIAL NEGATIVE SOCIAL IMPACT

Limitations of the work. Currently, DOTIN is more applicable to GAT-based models, as GAT learns edge weights by the network itself, making different virtual nodes target different tasks. While for pre-defined edges weights, multiple nodes learn the same embeddings, which is contradictory to our initial design and rationale. New edge pruning methods can be devised to address this issue.

Potential negative societal impacts. Deep learning can be time and energy-consuming. While our methods speed up GNNs and also with much less memory, yet DOTIN may make the future GNN community toward deeper/wider architectures, which may in turn increase energy consumption.

[1]We are a bit in short of confidence to rephrase the exact structure of the network for graph feature extraction in that 5-page arxiv paper (Pham et al., 2017), whereby the details are not well described, and no modern GNN structure, but instead RNN is used.