A Deep Hybrid Pooling Architecture for Graph Classification with Hierarchical Attention

Sambaran Bandyopadhyay[1,2(✉)], Manasvi Aggarwal[2], and M. Narasimha Murty[2]

[1] IBM Research AI, New Delhi, India
[2] Indian Institute of Science, Bangalore, Bengaluru, India
sambaran@alum.iisc.ac.in,{manasvia,mnm}@iisc.ac.in

Abstract. Graph classification has been a classical problem of interest in machine learning and data mining because of its role in biological and social network analysis. Due to the recent success of graph neural networks for node classification and representation, researchers started extending them for the entire graph classification purpose. The main challenge is to represent the whole graph by a single vector which can be used to classify the graph in an end-to-end fashion. Global pooling, where node representations are directly aggregated to form the graph representation and more recently hierarchical pooling, where the whole graph is converted to a smaller graph through a set of hierarchies, are proposed in the literature. Though hierarchical pooling shows promising results for graph classification, it looses a significant amount of information in the hierarchical architecture. To address this, we propose a novel hybrid graph pooling architecture, which finds the importance of different hierarchies of pooling and aggregates them accordingly. We use a series of graph isomorphism networks, along with a bi-directional LSTM with self attention to implement the proposed hybrid pooling. Experiments show the merit of the proposed architecture with respect to a diverse set of state-of-the-art algorithms on multiple datasets.

Keywords: Graph Neural Network · Hierarchical graph representation · Graph pooling · Self attention

1 Introduction

Graphs are important data types to represent different kinds of relational objects such as molecular structures, protein-protein interactions and information networks [12,16]. A graph is represented by $G = (V, E)$, where V is the set of nodes and E is the set of edges. Real-world graphs often come with a set of attributes, where each node $v_i \in V$ is also associated with an attribute (or feature) vector $x_i \in \mathbb{R}^D$. Graph classification, i.e., predicting the class label of an entire graph, is a classical problem of interest to the machine learning and data

K. Karlapalem et al. (Eds.): PAKDD 2021, LNAI 12712, pp. 554–565, 2021.
https://doi.org/10.1007/978-3-030-75762-5_44

mining community. Different practical applications such as finding anti-cancer activity, solubility, or toxicity of a molecule can be addressed by classifying the entire graph [9]. Graph classification is tackled as a supervised task. More formally, given a set of M graphs $\mathcal{G} = \{G_1, G_2, \cdots, G_M\}$, and a subset of graphs $\mathcal{G}_s \subseteq \mathcal{G}$ with each graph $G_i \in \mathcal{G}_s$ labelled with $Y_i \in \mathcal{L}_g$ (the subscript g stands for 'graphs'), the task is to predict the label of a graph $G_j \in \mathcal{G}_u = \mathcal{G} \setminus \mathcal{G}_s$ using the structure of the graphs and the node attributes, and the graph labels from $\mathcal{G}_s$. This leads to learning a function $f_g : \mathcal{G} \mapsto \mathcal{L}_g$. Here, $\mathcal{L}_g$ is the set of discrete labels for the graphs. Graph kernel algorithms [13,16] remained to be the stat-of-the-art for a long time for graph classification. Graph kernels typically rely on different types of hand crafted features such as the occurrence of some specific subgraph patterns in a graph.

Recently, graph neural networks (such as graph convolution network) and and node embedding techniques [1,4,7,17] are able to achieve promising results for node classification and representation. They map the nodes of the graph from non-Euclidean to Euclidean space by using both the link structure and the node attributes, and consequently use the vector representation for node classification in an end-to-end (or integrated) fashion. The main challenge to extend these approaches from node representation to graph representation is to design an intelligent node aggregation technique which can map a graph to a smaller graph (also known as graph pooling). There exist two types of graph pooling strategies in the literature. First, global pooling uses simple aggregation (such as averaging or concatenating) techniques on the embeddings of all the nodes to get a single vector representation of the graph [3]. Global pooling strategy works better for smaller graphs where aggregating all the nodes with a single function makes more sense. Second, hierarchical pooling recursively maps the input graph into a smaller graph (which may even contain a single node) and finally uses some aggregation technique [20].

Graphs exhibit hierarchical structures by default [2]. Nodes are the entities at the lowest of this hierarchy. Multiple nodes may form a sub-community and few sub-communities may for a community. In contrast to text documents where words, sentences and paragraphs form the hierarchy, hierarchical structure of a graph is latent in nature. Hierarchical graph neural networks, such as DIFFPOOL [20], jointly discover the hierarchical nature of the graph using graph pooling and finally convert it to a single node which is used to classify the entire graph. But different entities in the hierarchy do not play equal role to determine the label of a graph. For example, some intermediate level in the hierarchy may contain more useful information for classifying the graph rather than the final layer. This is true even for document classification, where importance of all the words and sentences are not the same to classify the entire document [19]. Besides, hierarchical pooling for graphs, though performs better, looses a significant amount of information in the multiple hierarchies of the pooling layers [10]. For example, if the graph after the final pooling layer contains a single node [20], then it is difficult for that node to encode all the structural information of the entire input graph in it.

To address the above challenges, we propose a novel graph pooling technique for graph classification, by mixing hierarchical and global pooling. Our algorithm (referred as *HybridPool*) maps the input graph to consecutive smaller graphs in multiple hierarchies (or levels) and then employs a global pooling across the multiple hierarchies (or levels). We observe that these graph hierarchies in the pooling network form a sequence. As the importance of different hierarchies are unknown, we use a bi-directional LSTM [5] with self-attention to weight them accordingly. Following are the contributions we make in this work.

- We propose a novel hybrid graph pooling algorithm *HybridPool* which employs multiple layers of graph convolution to create multiple hierarchies (or levels) of smaller graphs from an input graph, and then use bi-directional LSTM with self-attention to get the final representation of the entire graph. In contrast to existing literature, HybridPool learns different weights for different intermediate entities of a hierarchical representation of a graph and aggregates them globally for the entire graph classification. We use cross entropy loss of graph classification to jointly learn all the parameters of the entire network using back propagation.
- We conduct thorough experimentation on real-world graph datasets and achieve competitive performance with respect to state-of-the-art graph classification algorithms.

2 Proposed Solution: HybridPool

In this section, we describe the proposed algorithm HybridPool for graph classification.

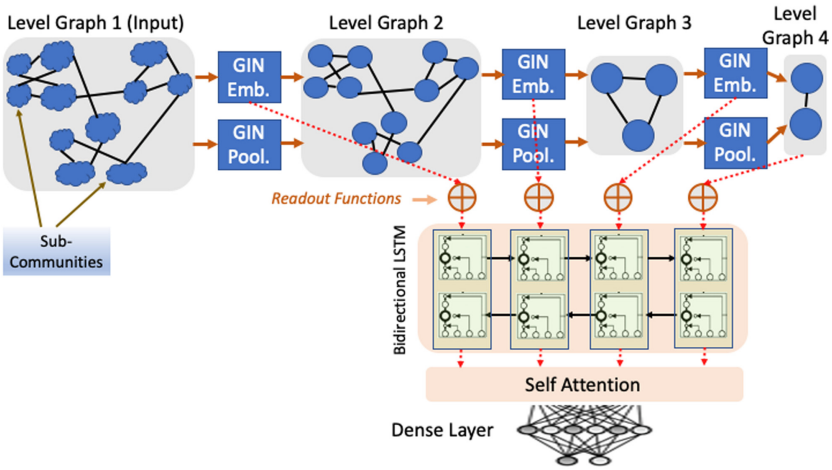

Fig. 1. Architecture of a HybridPool Network for graph classification

2.1 Overview of the Architecture

Before going to the details of the individual layers, we present a high level overview of HybridPool. Figure 1 shows the architecture of a HybridPool network with $R = 4$ level graphs. The first level is an input graph from the set of graphs $\mathcal{G}$. Next level graphs form different hierarchies in the graph, such as sub-communities, communities etc. Let us denote these level graphs (i.e., graphs at different levels) by $G^1, \cdots, G^R$. In level graph G^r, number of nodes is N_r, and the dimension of a feature vector for a node is K (except the input level graph G^1 which has D dimensional feature vector for each node). There is a GNN layer between level graph G^r (i.e., the graph at level r) and level graph G^{r+1}. This GNN layer comprises of an embedding layer which generates the embedding of the nodes of G^r and a pooling layer which maps the nodes of G^r to the nodes of G^{r+1}. A GIN [17] embedding layer and a pooling layer together convert a graph to a smaller (having lesser number of nodes) graph. We discuss the details of them in next two subsections. We refer the GNN layer between level graph G^r and G^{r+1} by rth layer of GNN, $\forall r = 1, 2, \cdots, R-1$. The last level graph G^R contains only one node, whose feature summarizes the entire input graph. Pleas note, number of nodes N_1 in the first level graph depends on the input graph, but we keep the number of nodes N_r in the consequent level graphs G^r ($\forall r = 2, \cdots, R$) fixed for all the input graphs (in a graph classification dataset), which helps us to build the next stages of the network conveniently as discussed below. Different level graphs can have different interpretation. In Fig. 1, each node in level graph 2 roughly represents a sub-community of the input graph, each node in level graph 3 can represent a community of the input graph, and finally the node in level graph 4 represents the whole input graph.

As mentioned in Sect. 1, the last level graph (for example, level graph 4 with two nodes in Fig. 1) is meant to encode all the structural and attribute information of the input graph at level 1. But as discussed before, due to the loss of some information in the formation of each level graph, the last level graph may not be able to preserve the intrinsic properties of the input graph to classify it. Besides, it is possible that the label information of the input graph depends directly on the overall sub-community structure or the community structure of the input graph, which are more prominent in some intermediate level graphs. So, we propose to learn the importance of different level graphs in the training phase. We use a bidirectional LSTM (BLSTM), which captures the ordered dependency of different level graphs. The use of bidirectional LSTM instead of a regular LSTM ensures the explicit modeling of a level graph as a function of the level graphs, both up and down in the hierarchy. First to obtain a summary vector for each level graph, we use a readout function. More precisely, a readout function takes the embeddings of all the nodes from a level graph and map them to a single vector which is invariant to the ordering of the nodes inside a level graph. The output of readout functions (one summary vector for each level graph as shown in Fig. 1) is fed to BLSTM. A self attention layer is used after the BLSTM layer to determine the importance of individual level graphs and aggregate them accordingly. Finally, this representation is fed to a dense

neural network with a softmax layer at the end to classify the entire graph. This completes one forward pass of the proposed HybridPool network.

Please note that the upper layers of HybridPool network resemble a hierarchical pooling structure (DIFFPOOL [20]) where an input graph is pooled to smaller graphs recursively. But there are crucial design differences between DIFFPOOL and the upper layers of the architecture adopted by us. First, DIFFPOOL uses GCN [7] as the embedding and pooling mechanisms. We use GIN [17] as the embedding and pooling mechanisms. The representation power of GIN is theoretically more than GCN. Second, DIFFPOOL always needs to have only a single node in the last level of the hierarchy to represent the whole graph by the features of that single node. But we may have more than one node in the last level, as the final graph representation is not obtained directly from the hierarchical structure. Rather, we use readout functions to compute a summary vector from each level graph and they are fed to BLSTM and the self-attention layers to compute the graph representation in HybridPool. Further, experimental comparison with DIFFPOOL, both for graph classification and model ablation study in Sect. 3 shows the merit of such a design adopted by us.

Hybrid Nature of the Pooling Strategy in HybridPool: The pooling strategy adopted by us is a mixture of hierarchical and global pooling strategies. The upper layers of HybridPool is a hierarchical graph pooling strategy, as discussed in the last paragraph. Lower layers which consist of the readout functions, bidirectional LSTM and self attention resemble a global pooling structure, as features from all the nodes in a level graph are directly aggregated and processed further to compute the final graph representation. That is why we call the proposed architecture *HybridPool*. Individual components of HybridPool are discussed below.

2.2 GIN Embedding Layer

This subsection defines the embedding layer (referred as GIN Emb. in Fig. 1) used in HybridPool algorithm. We use the GNN discussed in Graph Isomorphism Network (GIN) [17] as that is theoretically shown to be maximally powerful GNN to represent graphs, but still simple in nature. For a graph $G = (V, E)$, with adjacency matrix $A \in \mathbb{R}^{N \times N}$ and node attribute matrix $X \in \mathbb{R}^{N \times D}$, the l-th layer of GIN can be defined as:

$$h_v^{l+1} = MLP^l\Big((1 + \epsilon^l)h_v^l + \sum_{u \in \mathcal{N}(v)} h_u^k\Big) \tag{1}$$

Here, $h_v^{l+1} \in \mathbb{R}^K$ is the hidden representation of the node v in $l + 1$th layer of GIN and K is the feature dimension of the hidden layers of GCN. $\mathcal{N}(v)$ is the neighbors of the node v in G. ϵ is a parameter of GIN which determine the importance of a node's own representation with respect to the aggregated representation of its neighbors. The initial representations H^0 of the nodes are initialized with their respective features, $H^0 = X$.

For most of the datasets in Sect. 3, our GIN embedding layer consists of 1 or 2 layered deep GCN. The rth GIN embedding layer (between level graph r and $r+1$) is defined as:

$$Z_r = \text{GIN}_{r,embed}(A_r, X_r) \tag{2}$$

Here, $Z_r \in \mathbb{R}^{N_r \times K}$ is the final embedding matrix of the nodes of rth level graph G^r.

2.3 GIN Pooling Layer

We again use the same GIN proposed in [17] and as discussed in Sect. 2.2 as the GIN pooling layer (denoted as GIN Pool. in Fig. 1) of HybridPool. But the goal of pooling layer is to map the nodes from a previous level graph to next level graph, its output feature dimension is different from that of the embedding layer. Also we use a softmax layer after the final layer of GIN in the pooling layer as defined below.

$$P_r = \text{softmax}(\text{GIN}_{r,pool}(A_r, X_r)) \tag{3}$$

Here, (i,j)th element of $P_r \in \mathbb{R}^{N_r \times N_{r+1}}$ gives the probability of assigning node v_i^r in G^r to node v_j^{r+1} in G^{r+1}. The softmax in the pooling is applied row-wise.

2.4 Formulation of a Level Graph

Level graph 1 in the architecture of HybridPool is the input graph itself. This subsection discusses the formation of level graph G^{r+1} from G^r using the GIN embedding layer and the GIN pool layer. The adjacency matrix A_{r+1} of G^{r+1} is constructed as:

$$A_{r+1} = P_r^T A_r P_r \in \mathbb{R}^{N_{r+1} \times N_{r+1}} \tag{4}$$

Similarly, feature matrix X_{r+1} of G^{r+1} is constructed as:

$$X_{r+1} = P_r^T Z_r \in \mathbb{R}^{N_{r+1} \times K} \tag{5}$$

The matrix P_r contains information about how nodes in G^r are mapped to the nodes of G^{r+1}, and the adjacency matrix A_r contains information about the connection of nodes in G^r. Eq. 4 combines them to generate the link structure between the nodes (i.e., the adjacency matrix A_{r+1}) of G^{r+1}. Node feature matrix X_{r+1} of G^{r+1} is also generated similarly. Please note, G^1 is the input graph. For the intermediate level graphs, the number of nodes is determined by pooling ratio p as:

$$N_{r+1} = pN_r \ , \ \forall \ 1 \leq r \leq R-1 \tag{6}$$

Pooling ratio $p \in (0,1)$ is a hyper-parameter of HybridPool. We vary the pooling ratio from 0.05 to 0.5, depending on the size of input graph.

2.5 Attending the Important Hierarchies

In the above subsections, we discussed the generation of different level graphs from the input and their corresponding node features. As discussed in Sect. 1, graphs exhibit hierarchical structure and importance of different hierarchies are different to determine the label of the entire graph. So a simple global pool on all the nodes in the hierarchy may not be able to capture the varying importance across the levels of the hierarchy. This would become more evident on the model ablation study in Sect. 2.7.

In this subsection, we aim to determine the importance of different nodes in level graph 2 onward, for classifying the entire input graph. We employ a Bidirectional Long Short Term Memory Recurrent Neural Networks (BLSTM) and an attention layer for this purpose. BLSTM is proposed to handle the problem of vanishing gradients for recurrent neural networks [5] and they have been applied successfully to multiple NLP and sequence modeling tasks. In contrast to an LSTM, a bidirectional LSTM (or BLSTM) [21,22] processes data in both directions with two separate hidden layers, which are then fed forward to the same output layer. As shown in Fig. 1, we use a readout function to obtain the summary of a level graph. The readout function takes the GIN embeddings Z_r (from Eq. 2) of all the nodes of a level graph (except for the last level) and outputs a single vector. For the last level graph, there is no GIN embedding layer to generate the node embeddings. Also typically the number of nodes in the last level graph is really small. So instead of adding another GIN encoder, we use the readout function on the node feature matrix X_R (from Eq. 5) to generate the summary vector x^{G^R}. The readout function needs to be invariant to the ordering of the nodes in a graph. There are different types of readout functions proposed in the context of graph neural networks. They can be some simple aggregators such as sum (or mean) of the node embeddings [17], or attention-based node aggregators [10]. We use a simple readout function which just computes the sum of the node embeddings to obtain a summary vector of a level graph. This is because we further process those summary embeddings with the BLSTM and self-attention as discussed next. If we denote the summary vector of the level graph G^r as $s^r \in \mathbb{R}^K$, then

$$s^r = \begin{cases} \sum_{i=1}^{N_r} (Z_r)_{i,:} & 1 \leq r \leq R-1 \\ \sum_{i=1}^{N_R} (X_R)_{i,:} & \text{otherwise} \end{cases} \tag{7}$$

Though nodes within a level graph do not have any fixed order, but levels graphs themselves have an explicit hierarchical order. For example, a node can be a part of sub-community, the sub-community can be a part of a community, and so on. We use a BLSTM to explicitly capture this ordering of the level graphs. For each level graph G^r, $1 \leq r \leq R$, we feed its summary vectors s^r to the BLSTM as shown in Fig. 1. The BLSTM provides the forward-LSTM and backward-LSTM hidden representations h_r^f and h_r^b for each level graph G^r. We

concatenate and map them to K dimensional space as $h_r = \sigma(W_L[h_r^f||h_r^b])$, $\forall$ $1 \leq r \leq R$, where $W_L \in \mathbb{R}^{K \times 2K}$ and σ is an activation function. We discuss the attention layer next.

Attention mechanism has been used in node embedding in multiple works [8,15,18]. Here we propose to use a self attention mechanism over all the hidden representations $h_{r,i} \in \mathbb{R}^K$ to determine their importance in the final graph representation. Let us use $\mathcal{H}$ to denote the matrix containing each $h_{r,i}$ as a row. The self attention mechanism is defined below:

$$e = \text{softmax}(\mathcal{H}\theta),\ h = \mathcal{H}^T e \in \mathbb{R}^K \tag{8}$$

Here, $\theta \in \mathbb{R}^K$ is a trainable attention vector. Intuitively, each element of this vector determines the importance of a feature dimension of the node representations of the level graphs. So e in turn contains the normalized attention score (importance) of the individual nodes of level graphs from 2 onward. Finally, h is the final vector representation of the input graph, which is the sum of the representations of the nodes of the level graphs from 2 onward, weighted by their normalized attention score. The final graph representation h is fed to a dense neural network, followed by a softmax layer to classify the graph. Backpropagation algorithm with ADAM optimization technique [6] by minimizing the cross entropy loss of graph classification on the training set $\mathcal{G}_s$ is used to learn all the parameters of the architecture in an end-to-end fashion.

2.6 Run Time Complexity of HybridPool

There are different components of HybridPool. The runtime of a GIN depends on the size of the trainable parameter matrix and the number of nodes. For an input graph with N nodes, the total runtime for GIN embedding and pooling layers are $O(NDKR)$ and $O(NDN_2 + N\sum_{r=2}^{R-1} N_r N_{r+1})$ respectively, where D is the input node feature dimension, K is the dimension of final graph representation, R is the number of level graphs and N_r is the number of nodes in rth level graph. Computation of the summary vectors of all the level graphs takes $O(K\sum_{r=2}^{R} N_r)$ time. Next, Bidirectional LSTM and self attention layers take $O(KR)$ time. Hence, the runtime to process an input graph in HybridPool is $O(NDKR + NDN_2 + N\sum_{r=2}^{R-1} N_r N_{r+1} + K\sum_{r=2}^{R} N_r)$. For all the experiments, values of N_r, $r > 1$ and number of levels R are small. So, HybridPool is scalable even for large graph classification datasets. Also note that some of the steps above are easy to run in parallel, which can reduce the runtime further.

2.7 Variants of HybridPool: Model Ablation Study

HybridPool has multiple components and layered architecture. So, estimating the importance of individual components of this architecture is necessary. To give more insight about it, we present the following two variants of HybridPool.

- **MaxPool**: Here we replace the BLSTM and the attention layer of HybridPool with a maximum function $\max_{r>1,i}\{X_{r,i}\} \in \mathbb{R}^K$, which selects the maximum value for each feature dimension from all the nodes. Please note, $X_{r,i}$ denotes the K dimensional embedding of the ith node in level graph $r > 1$ (Eq. 5). In contrast to HybridPool, MaxPool has a static rule which selects the most important node for each feature dimension. $\max_{r>1,i}\{X_{r,i}\}$ is fed to the dense layer followed by a softmax.
- **BLSTMPool**: Here we remove the self attention layer from HybridPool. Instead, the concatenated hidden states from the final cells of the bidirectional LSTM is directly fed to the dense layer followed by a softmax for graph classification. Please note, BLSTMPool does not give different importance to different nodes in the level graphs.

It is to be noted that, above two architectures are simpler compared to HybridPool, as they miss one or more components from it. Thus, by comparing the performance of HybridPool with both of them in Sect. 3, we are able to show the significance of different components of HybridPool.

Table 1. Different datasets used in our experiments

Dataset	#Graphs	#Max Nodes	#Labels	#Attributes
MUTAG	188	28	2	NA
PTC	344	64	2	NA
PROTEINS	1113	620	2	1
NCI1	4110	111	2	NA
NCI109	4127	111	2	NA
IMDB-BINARY	1000	136	2	NA
IMDB-MULTI	1500	89	3	NA
REDDIT-M-12K	11929	3782	11	NA

3 Experimental Evaluation

In this section, we conduct thorough experimentation to validate the merit of HybridPool for graph classification. We also experimentally show more insights about the proposed architecture.

3.1 Datasets and Baselines

We use 5 bioinformatics graph datasets and 3 social network datasets to evaluate the performance of graph classification. These datasets are MUTAG, PTC, PROTEINS, NCI1, NCI09 and IMDB-MULTI. The details of these datasets can be found at (https://bit.ly/39T079X). Table 1 contains a high-level summary of

Table 2. Classification accuracy (%) of different algorithms (23 in total) for graph classification on 8 benchmark datasets. NA denotes the case when the result of a baseline algorithm could not be found on that particular dataset from the existing literature. The last row 'Rank' is the rank (1 being the highest position) of our proposed algorithm HybridPool among all the algorithms present in the table.

Algorithms	MUTAG	PTC	PROTEINS	NCI1	NCI109	REDDIT-M-12K	IMDB-B	IMDB-M
GK	81.39 ± 1.7	55.65 ± 0.5	71.39 ± 0.3	62.49 ± 0.3	62.35 ± 0.3	31.82 ± 0.08	NA	NA
RW	79.17 ± 2.1	55.91 ± 0.3	59.57 ± 0.1	NA	NA	NA	NA	NA
PK	76 ± 2.7	59.5 ± 2.4	73.68 ± 0.7	82.54 ± 0.5	NA	NA	NA	NA
WL	84.11 ± 1.9	57.97 ± 2.5	74.68 ± 0.5	**84.46 ± 0.5**	**85.12±0.3**	39.03	NA	NA
AWE-DD	NA	NA	NA	NA	NA	39.20 ± 2.09	74.45 ± 5.8	51.54 ± 3.6
AWE-FB	87.87 ± 9.7	NA	NA	NA	NA	41.51 ± 1.98	73.13 ± 3.2	51.58 ± 4.6
node2vec	72.63 ± 10.20	58.85 ± 8.00	57.49 ± 3.57	54.89 ± 1.61	52.68 ± 1.56	NA	NA	NA
sub2vec	61.05 ± 15.79	59.99 ± 6.38	53.03 ± 5.55	52.84 ± 1.47	50.67 ± 1.50	NA	55.26 ± 1.54	36.67 ± 0.83
graph2vec	83.15 ± 9.25	60.17 ± 6.86	73.30 ± 2.05	73.22 ± 1.81	74.26 ± 1.47	NA	71.1 ± 0.54	50.44 ± 0.87
InfoGraph	89.01 ± 1.13	61.65 ± 1.43	NA	NA	NA	NA	73.03 ± 0.87	49.69 ± 0.53
DGCNN	85.83 ± 1.7	58.59 ± 2.5	75.54 ± 0.9	74.44 ± 0.5	NA	41.82	70.03 ± 0.9	47.83 ± 0.9
PSCN	88.95 ± 4.4	62.29 ± 5.7	75 ± 2.5	76.34 ± 1.7	NA	41.32 ± 0.32	71 ± 2.3	45.23 ± 2.8
DCNN	NA	NA	61.29 ± 1.6	56.61 ± 1.0	NA	NA	49.06 ± 1.4	33.49 ± 1.4
ECC	76.11	NA	NA	76.82	75.03	41.73	NA	NA
DGK	87.44 ± 2.7	60.08 ± 2.6	75.68 ± 0.5	80.31 ± 0.5	80.32 ± 0.3	32.22 ± 0.10	66.96 ± 0.6	44.55 ± 0.5
DiffPool	NA	NA	76.25	NA	NA	47.08	NA	NA
IGN	83.89 ± 12.95	58.53 ± 6.86	76.58 ± 5.49	74.33 ± 2.71	72.82 ± 1.45	NA	72.0 ± 5.54	48.73 ± 3.41
GIN	89.4 ± 5.6	64.6 ± 7.0	76.2 ± 2.8	82.7 ± 1.7	NA	NA	75.1 ± 5.1	52.3 ± 2.8
1-2-3GNN	86.1 ±	60.9 ±	75.5 ±	76.2 ±	NA	NA	74.2 ±	49.5 ±
3WL-GNN	90.55 ± 8.7	**66.17 ± 6.54**	77.2 ± 4.73	83.19 ± 1.11	81.84 ± ± 1.85	NA	72.6 ± 4.9	50 ± 3.15
MaxPool	90.28 ± 4.98	63.00 ± 7.03	74.23 ± 3.10	80.33 ± 2.08	78.56 ± 2.00	43.12 ± 4.01	74.01 ± 3.08	47.73 ± 5.53
BLSTMPool	90.44 ± 5.09	61.92 ± 9.61	75.1 ± 2.93	81.10 ± 1.99	79.81 ± 1.21	45.99 ± 3.72	74.98 ± 3.64	50.60 ± 3.43
HybridPool	**92.02 ± 4.63**	65.99 ± 5.47	**78.34 ± 2.37**	82.76 ± 1.95	80.58 ± 2.15	**48.21 ± 3.61**	**76.39 ± 5.93**	**52.60 ± 4.01**
Rank	1	2	1	3	3	1	1	1

these datasets. We compare the performance of our proposed algorithms with a diverse set of state-of-the-art baseline algorithms for graph classification. The twenty baselines algorithms can broadly be classified into **three groups**: Graph Kernel Based Algorithms, Unsupervised Graph Representation Algorithms and Graph Neural Network based Algorithms. Additionally, we also show the classification results for the two simpler variants of HybridPool - MaxPool and BLSTMPool. Comparison to these variants experimentally show the marginal contribution of different components of HybridPool. To obtain the results of existing baseline algorithms on different graph classification datasets, we check the respective papers and state-of-the-art papers from the literature [11], and report the best classification accuracy available. Thus, we avoid any degradation of the performance of baselines due to insufficient parameter tuning.

3.2 Performance Analysis for Graph Classification

Table 2 shows the graph classification performance of all the baselines, our proposed algorithm HybridPool and its two simpler variants. We have grouped the algorithms according to their types in both the tables. We marked the places to be NA where the performance of some baseline algorithm is not present on a particular dataset for graph classification in the existing literature. Otherwise, all the results are obtained from the state-of-the-art research papers [11,17] and

best accuracy is reported when multiple of such results are present. As the number of algorithms presented in each of the tables are large in number, we also show the rank (1 being the best) of HybridPool among all the algorithms. For the bioinformatics graph datasets, HybridPool is able to improve the state-of-the-art for MUTAG and PROTEINS dataset. For PTC dataset, HybridPool is able to reach very close to 3WL-GNN, which is a very recently proposed graph neural network technique for graph classification. For NCI1 and NCI109, WL kernel consistently outperform other algorithms. For the social network datasets, HybridPool improves the state-of-the-art performance on all of REDDIT-M-12K, IMDB-B and IMDB-M. 3WL-GNN and GIN also perform well among the baselines.

Also, we can see that in most of the cases the HybridPool is able to outperform MaxPool and BLSTMPool. Moreover, the performance of BLSTMPool is better than MaxPool, except on PTC. Note that, MaxPool does not have the BLSTM layer and the self attention layer of HybridPool. Similarly, BLSTMPool does not have the self attention layer of HybridPool. Thus the relative performance of these three algorithms show the marginal positive contribution of both the BLSTM layer and the self attention layer to boost the performance of HybridPool. In addition, the overall standard deviation of accuracies of the HybridPool is at par or often less than many of the baseline algorithms. This shows the stability of HybridPool.

4 Discussion and Future Work

Graph neural network is an important area of research in machine learning and data mining. In this paper, we introduce a novel graph pooling strategy by combining global and hierarchical pooling techniques for a graph. Experimentally, we are able to improve the state-of-the-art performance on multiple graph datasets. Our framework can be extended in two ways. First, one can replace the GIN update rule inside the embedding and pooling layers of HybridPool with other types of node aggregation strategies [4,7]. Second, one can try different types of attention mechanisms [14] in our framework.

References

1. Bandyopadhyay, S., Lokesh, N., Murty, M.N.: Outlier aware network embedding for attributed networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 12–19 (2019)
2. Clauset, A., Moore, C., Newman, M.E.: Hierarchical structure and the prediction of missing links in networks. Nature **453**(7191), 98 (2008)
3. Duvenaud, D.K., et al.: Convolutional networks on graphs for learning molecular fingerprints. In: Advances in Neural Information Processing Systems, pp. 2224–2232 (2015)
4. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems, pp. 1025–1035 (2017)

5. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
6. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
7. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2017)
8. Knyazev, B., Taylor, G.W., Amer, M.: Understanding attention and generalization in graph neural networks. In: Advances in Neural Information Processing Systems, pp. 4204–4214 (2019)
9. Lee, J.B., Rossi, R., Kong, X.: Graph classification using structural attention. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1666–1674. ACM (2018)
10. Lee, J., Lee, I., Kang, J.: Self-attention graph pooling. In: International Conference on Machine Learning, pp. 3734–3743 (2019)
11. Maron, H., Ben-Hamu, H., Serviansky, H., Lipman, Y.: Provably powerful graph networks. In: Advances in Neural Information Processing Systems, pp. 2153–2164 (2019)
12. Morris, C., et al.: Weisfeiler and leman go neural: higher-order graph neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 4602–4609 (2019)
13. Shervashidze, N., Schweitzer, P., Leeuwen, E.J.V., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-Lehman graph kernels. J. Mach. Learn. Res. **12**(Sep), 2539–2561 (2011)
14. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
15. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018). https://openreview.net/forum?id=rJXMpikCZ
16. Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. J. Mach. Learn. Res. **11**(Apr), 1201–1242 (2010)
17. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=ryGs6iA5Km
18. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.I., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: International Conference on Machine Learning, pp. 5449–5458 (2018)
19. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., Hovy, E.: Hierarchical attention networks for document classification. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1480–1489 (2016)
20. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Advances in Neural Information Processing Systems, pp. 4800–4810 (2018)
21. Yu, Z., et al.: Using bidirectional LSTM recurrent neural networks to learn high-level abstractions of sequential features for automated scoring of non-native spontaneous speech. In: 2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), pp. 338–345. IEEE (2015)
22. Zhou, P., et al.: Attention-based bidirectional long short-term memory networks for relation classification. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (vol. 2: Short Papers), pp. 207–212 (2016)