EasyChair Preprint

№ 1056

Hashing Graph Convolution for Node Classification

Wenting Zhao, Zhen Cui, Chunyan Xu, Chengzheng Li,
Tong Zhang and Jian Yang

May 28, 2019

Hashing Graph Convolution for Node Classification

Anonymous Author(s)

ABSTRACT

Convolution on graphs has aroused great interest in AI due to its potential applications to non-gridded data. To bypass the influence of ordering and number of adjacent nodes, the summing/average diffusion/aggregation is often imposed on local receptive field in most prior works. However, the collapsing into one node in this way tends to cause signal entanglements of nodes, which would result in a sub-optimal feature information and decrease the discriminability of nodes. To address this problem, in this paper, we propose a simple but effective Hashing Graph Convolution (HGC) method by using global-hashing and local-projection on node aggregation for the task of node classification. In contrast to the conventional aggregation with a full collision, the hash-projection can greatly reduce the collision probability during gathering neighbor nodes. We argue the hash-projection based method can better preserve or even increase original discrepancies of local regions and further obtain improvement. Another incidental effect of hash-projection is that the receptive field of each node is normalized into a common-size bucket space, which not only staves off the trouble of different-size neighbors and their orders but also makes a graph convolution run like the standard shape-girded convolution. Considering the small training samples, also, we introduce a prediction-consistent regularization term into HGC to constrain the score consistency of unlabeled nodes in the graph. HGC is evaluated on both transductive and inductive experimental settings. The extensive experiments on node classification task demonstrate that hash-projection can indeed promote the performance and our HGC achieve new state-of-the-art results on all experimental datasets.

CCS CONCEPTS

• **Theory of computation → Semi-supervised learning**; *Machine learning theory*; Theory and algorithms for application domains;

KEYWORDS

Graph convolution, Node classification, Hash transform

ACM Reference Format:
Anonymous Author(s). 2019. Hashing Graph Convolution for Node Classification. In *Proceedings of The 28th ACM International Conference on Information and Knowledge Management (CIKM '18).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

CIKM '18, November 3-7, 2019, Beijing, China

ACM ISBN 978-1-4503-9999-9/18/06.
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

As a universal tool, graph has been widely-used to model various irregular data such as social networks, chemical molecules, recommendation systems and so on. Advanced by the powerful representation capabilities of standard convolution neural networks (CNNs) on shape-gridded data (e.g., image, video, etc.), the study of convolution on irregular data is getting increasing attention in the field of artificial intelligence. More recently, various methods [2, 11, 28, 31] start to flourish on graph convolution. However, due to the irregularity and complexity of geometric topology, generalizing CNN from regular grids to graphs is not a trivial thing. Since the adjacent vertices of one reference vertex are apparently orderless and with different quantities, the most existing works leverage a summing or averaging aggregation scheme on them. As the number of convolutional layers and local receptive fields increase, however, collapsing into one node in this way tend to cause excessive smoothing of the node features in a large range and lose discriminability information between nodes, which is also a reason that GCN [16] only employs the direct neighbors for feature aggregation. A simple example is shown in Figure 1. The vertices v_0 and v_6 in Figure 1(a) and Figure 1(b) are obviously different. After the averaging aggregation, their features will have the same result. That means, such a strategy might confuse those useful information when one benefits from its high-efficient computation. To address this problem, the hard sorting method of neighbor vertices was proposed in the literature [21], in which adjacent nodes are sampled into a fixed number and sorted in weights of edges. In view of the graph flexibility, the hard sorting is even sensitive to small disturbances. And redundant information may be incorporated in the transfer process.

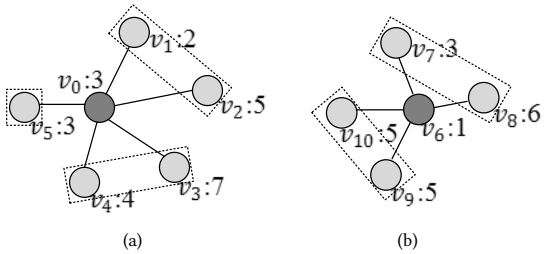
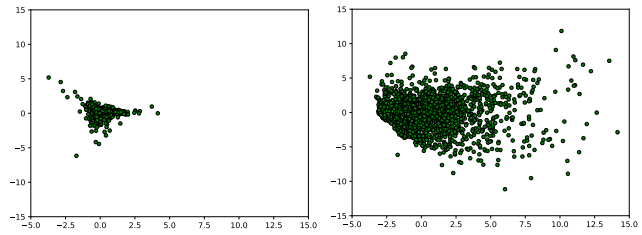


Figure 1: A toy example. The local receptive fields around v_0 and v_6 in (a) and (b) are obviously different. If the simple average aggregation is computed, i.e., $f(v_0) = (3 + 2 + 5 + 7 + 4 + 3)/6 = 4$, $f(v_6) = (3 + 6 + 10 + 5 + 5)/5 = 4$, their results are the same. But the hash-aggregation of three buckets are different with high probability, i.e., $f(v_0) = [(2 + 5)/2, (4 + 7)/2, 3, 3] = [3.5, 5.5, 3, 3]$ and $f(v_6) = [(5 + 5)/2, 0, (3 + 6)/2, 1] = [5, 0, 4.5, 1]$.



(a) average aggregation (b) hash-aggregation

Figure 2: Visualization of average aggregation *vs* hash-aggregation on Cora dataset. Only one time aggregation operation is imposed on the original features of one-hopping neighbors, and then we use principal component analysis to project the aggregated features into the first 2D space.

Based on the assumption that connected nodes should be similar, in essence, graph convolution is a smooth strategy, which aggregates the features of neighbors to the reference vertex. GCN [16] employs the simplest Laplacian smoothing with self-hop while GAT [28] takes advantage of weighted smoothing. Smoothing is the key of they both working well. The proper smoothing operation make features of nodes in same class similar, but over-smoothing would bring about features of nodes in different class indistinguishable. From this we can see that smoothing, as a main means of graph convolution, can improve the performance, but it also causes some useful information loss in the process. To remedy this defect, in this paper, we propose a simple and efficient Hashing Graph Convolution (HGC) method to encode graph vertices for node classification. Hash-projection is introduced to normalize vertices of each local receptive field, which should be the first time to our knowledge. Due to the good locality preservation property, hash-aggregation can better preserve the node discrepancies with high probability, compared to the conventional averaging aggregation. As shown in Figure 1, we randomly project nodes within receptive field into 3 hash buckets and then average features of all nodes in each bucket. The hash-aggregation results of v_0 and v_6 tend to be different with high probability. Also, this phenomenon is statistically observed in the real dataset as shown in Figure 2. In visualization of 2D principal component analysis, the distribution of vertices computed by hash projection in Figure 2(b) is significantly divergent, compared to the traditional aggregation in Figure 2(a). From the perspective of hashing, the prior average aggregation may be viewed as a particular case of ours with only a single bucket. In other words, the vertices within one receptive field fully collide into a bucket for the simple aggregation, and thus the local preservation ability of original feature space will be largely degraded. Multi-bucket hash-aggregation can effectively relieve this problem due to the good preservation of hash theory. Additionally, the probability of collision is proportional to the cosine distance of the two feature vectors, which means that nodes with the same label, in general, have similar features and are more easily projected into the same bucket. As a result, hashing also can avoid information redundancy by aggregating similar features into a bucket. Figure 3 exhibits the process of hash based aggregation and the detailed description for HGC is in Section 3.

Another incidental effect of hash-aggregation is that the local receptive field of each node is normalized into a common-size bucket space, which not only staves off the trouble of different-size neighbors and their orders but also make graph convolution analogical to the standard shape-girded convolution. The proposed hashing convolution can be easily incorporated into the prior convolution framework to boost the performance. Here, we consider the small portion of training vertices and introduce a regularization term of positive pointwise mutual information (PPMI) inspired by the recent work [34]. The regularization term constrains the entire graph structure by making the convolutional responses on the adjacent matrix as possibly consistent as the PPMI matrix. We assess our HGC model on both transductive and inductive experimental settings for node classification task. The extensive experiments demonstrate that hash-aggregation can indeed boost the performance of graph convolution, and meantime the advanced framework can benefit from this point. In transductive learning, HGC has broken through the general performance level of many methods and achieved the new state-of-the-art classification performance with only 20 samples in each class. At the same time, HGC also outperforms other methods and obtain superior performance in inductive inference, which testify that HGC can be generalized to unseen datasets

The remainder of this paper is organized as follows. Section 2 reviews some related works about graph convolution and hash transformation. Section 3 introduce our HGC model in detail and give the framework of the network. Section 4 describes the implementation details, reports the performance of and makes some discussions. Finally, Section 5 summarizes this paper and gives future research direction.

2 RELATED WORK

In this section, we briefly retrospect the related work including graph convolution and hashing methods.

Graph convolution is flourishing in the AI field recently. Niepert et al. [21] conducts traditional convolution operation by normalizing the graph to gridded structure, which converts neighborhoods of key nodes to fixed size as well as impose a order on them. In the process of transformation, some useful information may be lost. Then DGCNN [31] adds disordered graph convolutional layer(DGCL) based on mixed Gaussian model to avoid the loss of information. DCNN [2] presents a diffusion-convolution operation providing a straightforward mechanism for including K-hops neighborhoods information about each node. GraphSAGE [11] proposes a general inductive framework and applies several aggregator architectures to generate embeddings as input to downstream tasks. [28] introduces a masked self-attention layer that implicitly assigns different weights to different vertices in the neighborhood and obtains some improvement. GIC [14] leverages Gaussian mixture model (GMM) to encode local variations and conducts edge-Induced GMM and vertex-Induced GMM for convolution and pooling operation on graph, On the other hand, in the light of the Spectral Graph Theory [7], the spectral filtering based method has been successfully applied to the field of graphs. And in graph setting, the Laplacian eigenvalues provide a notion of frequency [25]. Convolution neural network (CNN) is firstly generalized to graph in [3] through

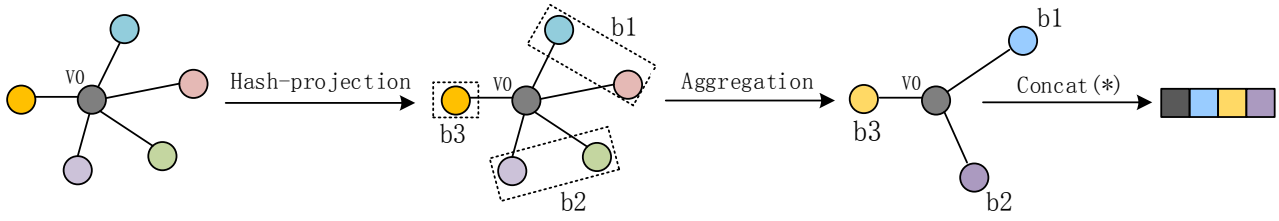


Figure 3: The process of hash based aggregation in 1-hop neighborhood. The neighbor nodes are hashed into 3 buckets, and then the node features in the 3 buckets are respectively aggregated, and the features in the reference node v_0 itself and the 3 buckets are concatenated together.

Laplacian eigenvalue decomposition, but the matrix computation is expensive and it is non-localized filters. To overcome these problems, the work [12] attempts to spatially localize through parameterizing the spectral filters with smooth coefficients. On this basis, ChebyNet [9] realizes fast localized spectral filter using Chebyshev polynomial approximation, which significantly reduces the computational complexity. Further, GCN [16] simplifies the filter to a linear function of first-order in the case of ensuring that the model ability isn't declined. Lately literatures [6, 18, 19, 34] have made some improvements on these foundations and obtain some gains, in which FastGCN [6] interprets GCN from the perspective of integral transformation and further accelerate GCN by samples vertex in each layer. And DGCN [34] devises a dual graph convolutional neural network method to jointly consider local and global information.

Hashing is mainly leveraged for feature dimensionality reduction and information retrieval. Feature Hashing [1] applies a hash transform to reduce dimension for collaborative filtering of spam, give an unbiased estimate formulation of the hash kernel. The work [8] constructs a sparse projection matrix through hashing and local densification and gives a matching lower bound on the sparsity for a large class of projection matrices. Hashing significantly improved the efficiency of calculation by means of a sparse projection into a lower dimensional space for very high dimensional setting [24]. In addition, hash transformation is widely used in the similarity search field due to its huge search cost. Due to similarity preserving [30], learning to hash has been applied to a wide-range of applications such as large scale object retrieval [13], image classification [23] and detection [27], and so on. In this work, our purpose is not to study concrete hashing methods, but introduce hashing into graph convolution, which should be first time to our knowledge.

3 OUR APPROACH

Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ denote an undirected/directed graph, where $\mathcal{V}$ represents a set of vertices with the number $|\mathcal{V}| = n$ and $\mathcal{E}$ is a set of edges. According to the adjacency relation in $\mathcal{E}$, we can define the corresponding adjacent matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ of the graph $\mathcal{G}$. If $(v_i, v_j) \in \mathcal{E}$, we set $A_{ij} = 1$, otherwise $A_{ij} = 0$. When edges has different weights, A_{ij} may be assigned to a real value. Besides, each vertex might have a feature description with a d-dimension vector $\mathbf{x} \in \mathbb{R}^d$. At this time, the features of all vertices, i.e., the graph features, could be stacked by rows into a feature matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$. To state conveniently, we use $\mathbf{X}_{i\cdot}$ or $\mathbf{x}_i$ to denote the feature of the i-th vertex.

3.1 Hashing Graph Convolution

Given the node set $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ of a graph, we seek for the hash functions $h(\cdot)$ to project the nodes into b different hash buckets $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_b\}$. We expect the hash functions can map those similar vertices into the same bucket with high probability, which means good local preservation ability. Afterwards, we need to represent local receptive fields of the irregular graph. In the standard convolution performing on images, the receptive field may be sophisticatedly defined as a local square space region, so convolution filtering on regular structures are operable easily. In contrast, each vertex of graph usually contains different neighbor numbers and these neighbor nodes are unordered apparently. Due to the high-degree flexibility of graph, one often uses the neighborhoods to define receptive field. Formally, given a reference vertex v_i, we can infer all its s-hopping reachable vertices, and denote the vertex set as $\mathcal{N}^s(v_i)$. When $s = 2$, the vertex set $\mathcal{N}^2(v_i)$ consists of the one-nearest neighbors starting v_i, i.e., the direct adjacent relationship. If $s = 1$, we let $\mathcal{N}^1(v_i) = \{v_i\}$. Here we call the vertex set $\mathcal{N}^s(v_i)$ as the s-scale receptive field. Moreover, the receptive field could be recursively derived as follows

$$\mathcal{N}^{s+1}(v_i) = \{v_l | v_j \in \mathcal{N}^s(v_i) \text{ and } v_l \in \mathcal{N}^1(v_j)\}, \tag{1}$$

where $s = 1, 2, \cdots, S$ and S is the maximum scale of receptive field. In the above definition of receptive fields, the intersection of two receptive fields might be nonempty, i.e., $\mathcal{N}^s \cap \mathcal{N}^{s+1} \neq$ null. One may take their set difference, $\mathcal{N}^{s+1} \leftarrow \mathcal{N}^{s+1} - \mathcal{N}^s$. In fact, it is not necessary for this according to our experiences.

Next, we induce the total formulation of hashing convolution filtering on one center vertex v_i, formally,

$$C * \mathcal{G}(v_i) = f(\mathbf{x}_i; \widetilde{\mathbf{x}}_i^2; \cdots; \widetilde{\mathbf{x}}_i^S), \tag{2}$$

$$s.t., \quad \widetilde{\mathbf{x}}_i^s = g([\widetilde{\mathbf{x}}_{i\to\mathcal{B}_1}^s; \cdots; \widetilde{\mathbf{x}}_{i\to\mathcal{B}_b}^s]), \tag{3}$$

$$\widetilde{\mathbf{x}}_{i\to\mathcal{B}_k}^s = F_{i\to\mathcal{B}_k}^s \sum_{\substack{v_j\in\mathcal{N}^s(v_i),\\ h(v_j)\in\mathcal{B}_k}} w_{ij}^s \mathbf{x}_j, \tag{4}$$

$$s = 2, \cdots, S; \; k = 1, 2, \cdots, b.$$

where w_{ij}^s is the weight between vertex v_i and v_j at the s-scale receptive field, f, g are non-linear functions to be solved, $[\cdot\,; \cdots\,; \cdot]$ means the concatenation of features, and $F_{i\to\mathcal{B}_k}^s$ is the normalization factor.

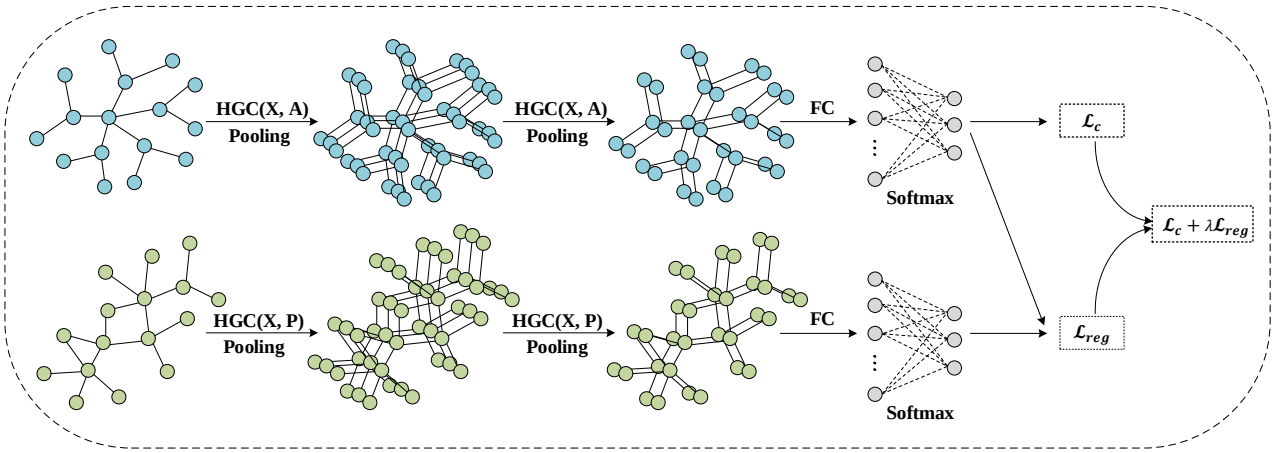


Figure 4: The overall framework of HGC-based network.

Term $\widetilde{\mathbf{x}}^s_{i\to\mathcal{B}_k}$: in the above Eqn. (4), it aggregates all the s-hopping neighbor vertices falling into the k-th hash buckets w.r.t. the reference v_i. Different from the convention aggregation, the hash-aggregation partitions those vertices within a receptive field into b different buckets. According to the Johnson-Lindenstrauss lemma [15], the embedded hashing projection on the vertex set can preserve up to a certain multiplicative factor of the original space. The detailed definition of the hash function $h(\cdot)$ will be introduced in Section 3.3. It is worth noting that the hash function is globally defined for all vertices of the graph but the hash-aggregation is only imposed on a few neighbor vertices. The global definition makes sure the hash-projection consistency of local convolution on different reference vertices. The local hash-aggregation may reduce the collision probability due to the number of neighbors is small (i.e., the connection edges are usually sparse). The weighting scalar w^s_{ij} may be chosen as the s-hopping reachable probability of random walks from vertex v_i to vertex v_j. In a simple way, we may also set $w^s_{ij} = 1$ if v_i and v_j just satisfies the s-hop reachability, otherwise 0. The factor F is used to eliminate effect of different hash-projected vertex numbers, and is defined as

$$F^s_{i\to\mathcal{B}_k} = 1/\left|\{v_j | v_j \in \mathcal{N}^s(v_i) \text{ and } h(v_j) \in \mathcal{B}_k\}\right|. \tag{5}$$

Term $\widetilde{\mathbf{x}}^s_i$: in Eqn. (3), it collects all buckets's features at the s-scale receptive field. Here, we concatenate them and then project into a low-dimension feature space by using the function g. Concretely, the function $g : \mathbb{R}^{d\times b} \to \mathbb{R}^d$ is defined as a fully-connected layer with the non-linear activation unit ReLU. Other strategies may be selected for this function.

Function f: in Eqn. (2), it maps the features of S receptive fields (include the reference vertex itself) to a low dimension space. Similar to the function g, we concatenate them and then project into a pre-specified dimension space by using a fully-connected layer and the ReLU unit.

3.2 Pooling

In node classification, we do not need to massacre vertices, i.e., graph coarsening. That means, the pooling should be vertex-wise diffusion on local region, such that the high-level semantic information can be abstracted from the graph. Formally, we perform the pooling on the S-scale receptive field w.r.t. one vertex v_i as

$$\mathcal{P}(\mathcal{G}(v_i)) = \mathcal{P}(\{\mathbf{x}_j | v_j \in \mathcal{N}^s(v_i), s = 1, \cdots, S\}), \tag{6}$$

where the pooling operation $\mathcal{P}$ is usually defined as "max" or "mean". In practice, their performances have little difference on graph convolution, but the "mean" pooling is more stable than "max". Thus, here we choose the average operation on all neighbor vertices during pooling. The detailed HGC network prediction process including pooling operation has been explained in Algorithm 1.

3.3 Hash Learning

For the hash function $h(\cdot)$, we attempt to learn the dynamic hashing by using data-dependency. Those similar features should be projecting into the same bucket with high probability. That means, the locality property in the original feature space will be largely preserved in the transformed space. Concretely, we build hash-projection on the features of vertices. Formally, the hashing function consists of b projecting parameters $\{\mathbf{w}_1, \cdots, \mathbf{w}_b\}$ and the biases $\{c_1, \cdots, c_b\}$,

$$h(\mathbf{x}) = \mathbf{argmax}_k\{\mathbf{w}_1\mathbf{x} + c_1, \cdots, \mathbf{w}_k\mathbf{x} + c_k, \cdots, \mathbf{w}_b\mathbf{x} + c_b\}, \tag{7}$$

where $\mathbf{w}_k$ may be understood as the hyperplane. When taking random projection, the hash function $h(\cdot)$ usually satisfies the locality preserving property, $P(h(\mathbf{x}_i) = h(\mathbf{x}_j)) \propto \exp\{-\frac{\|\mathbf{x}_1-\mathbf{x}_2\|^2}{\sigma^2}\}$, according to the theory of local sensitive hashing [10]. Due the asymptotic theoretical guarantees for hash projection, the above hash-aggregation could be preserve the discrepancy well. In contrast to the straightforward average aggregation collapsing all vertices within a receptive field into a vertex, the hashing convolution encodes the discrepancy of vertices better. Moreover, based on the hashing theory [29], the collision probability for two vertices is given by

$$P(h(\mathbf{x}_i) = h(\mathbf{x}_j)) \propto \left[1 - \frac{cos^{-1}(\mathbf{x}_i^T\mathbf{x}_j)}{\pi}\right]^b. \tag{8}$$

the probability of collision is proportional to the cosine distance of the two feature vectors, which means that nodes with the same label, in general, have similar features and are more easily projected into the same bucket. As a result, hashing also can avoid information redundancy by aggregating similar features into a bucket. With the increase of value of b, the false collisions will be reduce largely. In other words, those non-neighbor or dissimilar vertices falls into the same bucket with a lower probability. However, a large b also decreases the collision probability of those similar vertices. More importantly, for one local convolution, the filtered vertex number is far less than the total number of graph vertices due to the sparsity. Thus, the number b of buckets may be chosen a small value to achieve the satisfactory performance. In the hashing learning, we also put these parameters $\{\mathbf{w}_k, c_k\}$ into the entire network and jointly optimize them.

3.4 Loss with Regularizer

In the graph $\mathcal{G}$, given the training and validation node sets $\mathcal{V}_{tr}, \mathcal{V}_{val} \in \mathcal{V}$, whose vertices have been labeled, the aim is to estimate those unlabeled vertices of the test set $\mathcal{V}_{te} \in \mathcal{V}$. The validation set is used to tune the model parameters, when learning on the training set. Suppose the final convolution output $\mathbf{Y} \in \mathbb{R}^{n \times c}$ is formally

$$\widehat{\mathbf{Y}}^A = \text{conv}(\mathcal{V}, \mathbf{X}, \mathbf{A}, \Theta), \tag{9}$$

where the inputs consist of graph vertices $\mathcal{V}$, graph feature $\mathbf{X}$ and adjacency matrix $\mathbf{A}$, and Θ is the model parameters to be learnt. Then we use the cross-entropy loss only on the training set $\mathcal{V}_{tr}$,

$$\mathcal{L}_c = \frac{1}{|\mathcal{V}_{tr}|} \sum_{v_i \in \mathcal{V}_{tr}} (Y_{ij} == 1) \ln \widehat{Y}^A_{ij}. \tag{10}$$

The hyperparameters of our model are decided by the validation set $\mathcal{V}_{val}$.

However, a main limit is that a small portion of vertices are annotated as the training set. Thus, in the semi-supervised scenario, our aim is to use labeled vertices as well as graph structure to train model for good generalization ability. The straightforward way is using the graph Laplacian matrix $Ł = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ as a regularization term, which utilizes the implicit assumption that connected vertices in graph are likely to have the same label. However, this assumption has been actually used in the above graph convolution by neighbor aggregation. Also, as argued in the literature [34], graphic edges need not reflect the similarity of nodes, instead they might contain additional information. To this end, we employ a global consistency constraint through positive pointwise mutual information (PPMI) [4, 17], as used in [34]. Giving any one reference vertex v_i, the co-occurrence frequency F_{ij} w.r.t. vertex v_j may be calculated through random walks with the transition probability $p(s(t+1) = v_j | s(t) = v_i) = A_{ij} / \sum_j A_{ij}$. After obtaining the frequency matrix $\mathbf{F} \in \mathbb{R}^{n \times n}$, we derive the PPMI matrix $\mathbf{P} = \mathbf{max}\{\log\left(\frac{\mathbf{F}/(\mathbf{1}^T\mathbf{F}\mathbf{1})}{\mathbf{F}/(\mathbf{F}\mathbf{1}\mathbf{1}^T) \odot \mathbf{F}/\mathbf{1}\mathbf{1}^T\mathbf{F}}\right), 0\}$, where the division and log operations are the elementwise calculation. P_{ij} represents the estimated probability of vertex v_i occurred in the context $P_{\cdot j}$.

Since the PPMI matrix $\mathbf{P}$ indicates the co-occurrence relationships of nodes, we may take it as the weight matrix to feed into the hashing convolution in Eqn. (2)~(4). That is, we may initialize w_{ij} in Eqn.(4) with the PPMI vaule P_{ij} and then perform graph convolution. Formally, we can obtain the final convolution output: $\widehat{\mathbf{Y}}^P = \text{conv}(\mathcal{V}, \mathbf{X}, \mathbf{P}, \Theta)$. We use the output $\widehat{\mathbf{Y}}^P$ to specifically constrain those unlabeled vertices, so that $\widehat{\mathbf{Y}}^A \simeq \widehat{\mathbf{Y}}^P$ as used in [34]. Therefore, the optimized objective function with the regularization term is formulated as

$$\mathcal{L} = \mathcal{L}_c + \frac{\lambda}{|\mathcal{V}|C} \sum_{v_i \in \mathcal{V}} ||\widehat{\mathbf{Y}}^A_{i\cdot} - \widehat{\mathbf{Y}}^P_{i\cdot}||, \tag{11}$$

where λ is the balance factor between the labeled training vertices and graph structure. C is the number of classes. Note that $\widehat{\mathbf{Y}}_{i\cdot}$ is a probability vector of the vertex v_i after a softmax operation.

The framework of HGC employed by experiments is shown in Figure 4. Due to the existence of the regularization term, our framework is a dual stream network. As we discussed above, the adjacency matrix $\mathbf{A}$ and PPMI matrix $\mathbf{P}$ characterize different graph topology structure respectively. In other words, the connection of the graph is different, so two different structures of the graph are input into the network. The network is consisted with two hashing graph convolution layers followed by pooling layer and one fully connected layer with a softmax layer. An important difference from work [34] is that our dual stream network does not share network parameters.

Algorithm 1: HGC for node classification algorithm

Input: node features matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$; adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$; receptive fild scale S; the number of layers L; non-linearity functions f, g; hash function h;
Output: the predicted labels of nodes $Y \in \mathbb{R}^{n \times c}$;

for $v \in \mathcal{V}$ **do**
  $\mathcal{B}_k \leftarrow \mathbf{argmax}\{h(\mathbf{x}_v)\}$;
end
for $v_i \in \mathcal{V}$ **do**
  $\mathbf{z}_i^0 = \mathbf{x}^0$;
  for $l = 1, 2, \cdots, L$ **do**
    $\mathbf{x}_i = \mathbf{z}_i^l$;
    for $s = 1, 2, \cdots, S$ **do**
      for $k = 1, 2, \cdots, b$ **do**
        $\widetilde{\mathbf{x}}^s_{i \to \mathcal{B}_k} = \mathbf{aggregation}(\mathbf{x}_j)$;
        s.t. $v_j \in \mathcal{N}^s(v_i), h(v_j) \in \mathcal{B}_k$;
      end
      $\widetilde{\mathbf{x}}^s_i = g([\widetilde{\mathbf{x}}^s_{i \to \mathcal{B}_1}; \cdots; \widetilde{\mathbf{x}}^s_{i \to \mathcal{B}_b}])$
    end
    $\mathbf{z}_i^l = f(\mathbf{x}_i; \widetilde{\mathbf{x}}^2_i; \cdots; \widetilde{\mathbf{x}}^S_i)$;
    $\mathbf{z}_i^l = \mathbf{Pooling}\{\mathbf{Concat}(\mathbf{z}_i^l)\}, v_j \in \bigcup_s^S \mathcal{N}^s(v_i)$;
  end
  $Y_i = \mathbf{argmax}\{\mathbf{softmax}\{\mathbf{fc}(\mathbf{z}_i^L)\}\}$
end

4 EXPERIMENTS

In the section, we carry out extensive experiments and assess the performance of our HGC model on both transductive and inductive

Table 1: Summary of graph datasets used in experiments.

Dataset	Graphs	Nodes	Edges	Features	Classes	Label rate	Type	Task
Cora	1	2708	5429	1433	7	0.052	Citation	Transductive
Citeseer	1	3327	4732	3703	6	0.036	Citation	Transductive
Pubmed	1	19717	44338	500	3	0.003	Citation	Transductive
NELL	1	65755	266144	61278	210	0.008	Knowledge Graph	Transductive
PPI	24	56944	818716	50	121(multi-label)	-	Protein-Protein Interaction	Inductive

settings for node classification task. Transductive learning is the general semi-supervised learning, which training samples and test samples share graph topology structure. Inductive learning is generalizing the training model to unseen samples, i.e. test samples and training samples are on different graphs. The experimental results demonstrated our HGC model indeed outperforms other methods. We first outline the datasets and experimental setups, then compare our results with state-of-the-arts and make some analysis, after which ablation study will be held.

4.1 Datasets

The global properties of all datasets have been summarized in Table 1, and details are as follows.

- Citation graph. The Cora dataset consists of 2708 machine learning papers divided into one of seven classes. each node represents a document, and node features are bag-of-words representation of documents indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words. If an article cites another article, an undirected link/edge in added between them, a total of 5429 edges. Similar to the Cora dataset, Citeseer contains 3327 papers and every one belongs to one of six classes. Node features are also bag-of-words representation with 3037 unique words. There exists 4732 edges between the nodes. Also, as citation network, Pubmed is a larger dataset containing 19717 papers and 44338 edges. Instead of binary value, the node features have real-values entries indicating Term Frequency-Inverse Document Frequency (TF-IDF) of the corresponding word from a dictionary. We adopt the dataset preprocessed in work [33], and follow its data partitioning rules. There are only 20 samples in each class for training. A total of 500 samples are used for validation and 1000 samples are used for testing. Label rate has been reported in Table 1.
- Knowledge graph. NELL is a dataset extracted from the knowledge graph introduced in [5]. Node are connected by directed, labeled relation edges. Every edge is described by a triplet (e_h, r, e_t), e_h and e_t represent two entities/nodes, r denotes the relation/edges between them. We follow the pre-processing scheme as presented in [16, 33]. The triplet (e_h, r, e_t) is splitted into two edges (e_h, r_1) and (r_2, e_t). Finally, a graph with 65766 nodes and 266144 edges is obtained. The node features are extended by unique sparse one-hot representation. According to the work [34], 500 samples constitute training set. 105 and 969 samples are used for validation and test set.
- PPI. PPI is a protein-protein interaction dataset including 24 graphs, each corresponds to a human tissue [35]. The task is classifying protein functions according to gene ontology. We perform the same preprocessed method as work [11, 28]. Each node has 50 features contain information about motif gene sets, positional gene sets and immunological signatures and gene ontology sets are treated as labels, collected from the Molecular Signatures Database [26]. The dataset is exploited in inductive inference and is a multi-label problem. 20 graphs, 2 graphs and 2 graphs are used for training, validation and testing respectively.

4.2 Experimental Setups

Transductive learning. For the transductive learning, a two-layer HGC model is applied as other baseline methods for fairly comparison. Each convolution layer is followed by a pooling layer, and then outputs a predictive result through a fully connected layer and an output layer with softmax. The network structure can be simply represented as $Input - \mathcal{C}(128) - \mathcal{P}(mean) - \mathcal{C}(256) - \mathcal{P}(mean) - \mathcal{FC}(128) - Output(softmax)$, where $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{FC}$ denote convolution, pooling and fully connected layer. "128" and "256" represent the number of channels output by convolution or fully connected layer. $\mathcal{P}(mean)$ represents the "*mean*" operation used in the pooling layer. Dropout rate is set to 0.5 in the convolution and full connected layers, and rectified linear unit (ReLU) unit is leveraged as nonlinear activation function thereof. The receptive filed is set to 3 for citation and NELL datasets. We adopt Momentum optimizer to train HGC with 500 epochs, learning rate of 0.05, and momentum of 0.9. The regularizer coefficient λ is a temporal weighted function. As $\mathcal{L}$ is much larger than $\mathcal{L}_{Reg}$, the maximum is set as around 100.

Inductive learning. For inductive learning , we follow the settings in GAT [28]. A three-layer HGC model is applied. The network structure can be simply represented as $Input - \mathcal{C}(128) - \mathcal{P}(mean) - \mathcal{C}(256) - \mathcal{P}(mean) - \mathcal{C}(512) - \mathcal{P}(mean) - Output(sigmoid)$. Each convolution layer is followed by an exponential linear unit (ELU) nonlinearity function. Due to the large training set and graph structure is not shared among training and test set, thus the dropout rate is set to 0. The receptive filed is set to 2. In order to be able to converge faster, we adopt Adam optimizer to train the model for 800 epochs, learning rate is also 0.05. The batch size is set to 2. Since the structure is different between testing set and training set, a small weight with 0.01 is given to $\mathcal{L}_{Reg}$. This task on PPI dataset is a

multi-label classification with 121 labels, and sigmoid activation is utilized in output layer. The micro-F1 score is computed as accuracy.

The buckets number b is in the range $(5, 8)$ for all datasets. In Section 4.4, we perform ablation study about the convolution layer number L and the receptive filed S, and some analysis have been made on how to choose the regularizer coefficient λ and the number of buckets b for the hash-projection.

Table 2: Comparisons with state-of-the-art methods for transductive setting.

Method	Cora	Citeseer	Pubmed	NELL
DeepWalk [22]	67.2%	43.2%	65.3%	58.1%
Planetoid [33]	75.7%	64.7%	77.2%	61.9%
Chebyshev [9]	81.2%	69.8%	74.4%	-
GCN [16]	81.5%	70.3%	79.0%	66.0%
MoNet [20]	81.7%	-	78.8%	-
GAT [28]	83.0%	72.5%	79.0%	-
DGCN [34]	83.5%	72.6%	80.0%	74.2%
HGC(Ours)	**85.2%**	**74.3%**	**81.5%**	**78.0%**

4.3 State-of-the-art Comparison.

Transductive learning. For transductive setting, we compare the performance of our HGC model against a several of the state-of-the-art works: DeepWalk [22], Planetoid [33], Chebyshev [9], GCN [16], MoNet [20], GAT [28] and DGCN [34]. Note all these results coming from the related literatures. Table 2 reports the corresponding results, which clearly indicate that our HGC approach achieve the new state-of-the-art performance and obtain a remarkable improvement on these four datasets. DeepWalk and Planetoid predict the node's label by the way of generating the node embeddings, which is less accurate than other methods using graph convolution. GCN is a first-order approximation of Chebyshev and has realized relatively high results. Compared to GCN, our HGC method attains the results of 85.2% *vs* 81.5% on Cora dataset, and 74.3% *vs* 70.3% on Citeseer dataset, which are 3.7% and 4.0% higher than GCN. Also, 2.5% and 3.9% are raised on the Pubmed and NELL dataset. We attribute this gain to the hash-aggregation, due to the good locality property, which can better preserve the node discrepancies with high probability in contrast to the conventional averaging aggregation. Meanwhile, the regularizer item also play a key role. MoNet is a general framework employing Gaussian mixture model, GCN can be regarded as a special case of it, and their classification accuracy is similar. GAT and DGCN utilize the attention mechanism and global information respectively based on the basis of GCN, and both lead to performance gains. Compared to GAT, the HGC model still achieves superior performance and obtains 2.2%, 1.8% and 2.5% improvements on Cora, Citeseer and Pubmed datasets respectively. Although our HGC draws on the global consistency constraint used in DGCN through the positive pointwise mutual information matrix, there is still a marked improvement in our outcomes in contrast to DGCN. Specifically, 85.2% *vs* 83.5% on Cora dataset, 74.3% *vs* 72.6% on Citeseer dataset, 81.5% *vs* 80.0% on Pubmed dataset and 78.0% *vs* 74.2% on NELL dataset. In particular, HGC is 3.8% higher than DGCN on NELL dataset. This may be ascribed to the large categories number and the high degree of some nodes, which is suitable for hash-projection to play to its advantages. This is a strong evidence that hash-projection is an effective aggregation method, which keep the discriminative information between different classes of nodes as much as possible while smoothing similar nodes. Notably, the performance of some recent works have been at a similar level, and our HGC method has improved by or so 2% in the case of using only 20 training samples per class, which further verifies the effectiveness of HGC.

Table 3: Comparisons with state-of-the-art methods for inductive setting. The number in parentheses $(*)$ denotes the number of convolution layer in the network.

Method	Micro-F1
Random [11]	39.6%
MLP [11]	42.2%
GraphSAGE-GCN [11]	50.0%
GraphSAGE-mean [11]	59.8%
GraphSAGE-LSTM [11]	61.2%
GraphSAGE-pool [11]	60.0%
JK-Dense-LSTM (2L) [32]	97.6%
JK-Dense-LSTM (3L) [32]	96.9%
GAT [28]	97.3%
HGC (2L)	98.2%
HGC (3L)	**98.6%**

Inductive learning. For inductive learning, we mainly compare HGC with JK-Net [32], GAT [28] and GraphSAGE [11] in the case of supervision. GraphSAGE [11] performs sampling and aggregation operations at each layer to generate node embeddings as input to downstream tasks. GAT [28] assigns a weight to each neighbor of a node to measure their contribution during the aggregation process. JK-Net [32] uses a jump connection structure and adaptive neighborhood aggregation scheme. The other two baselines introduced in GraphSAGE [11]: Random (random classifer), MLP (logistic regression feature-based classifier). Due to the multi-label classification problem, micro-F1 score is calculated to measure the performance of the model, and the results are reported in Table 3. We can observe that HGC, JK-Net and GAT a significant improvement over GraphSAGE. One possible reason may be that the network structure is different according to GAT. To fairly evaluate the benefits of hash-projection, we employ the same number of convolution layer as GAT, i.e. three layers with [128, 256, 512] features in each layer. At the same time, we also apply two-layer HGC model with [256, 512] features computed in each layer to compare with JK-Net. The receptive field of both settings are set to 2 due to memory limitations. As shown in Table 3, HGC can be able to improve by a margin of 1.3% w.r.t. GAT. The result of HGC is still higher than JK-Net with a relatively small margin. The

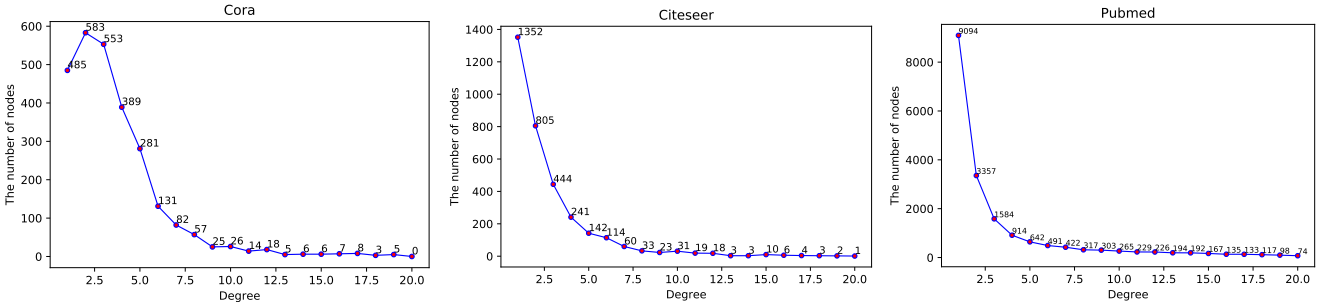


Figure 5: The distribution of degree on citation datasets.

excited gains indicate that the hash-projection is also effective for the inductive setting and can work well when structure information is not involved in the training set. The HGC model has the potential to be extend to more general situation.

4.4 Ablation Study

In this section, we explore and analyze how different hyperparameter settings affect classification accuracy. We mainly analyze from the following aspects:

- the number of convolution layers L.
- the size of receptive field S.
- the number of hash buckets b and the value of regularizer coefficient λ.
- the effect of hash-projection on model ability.

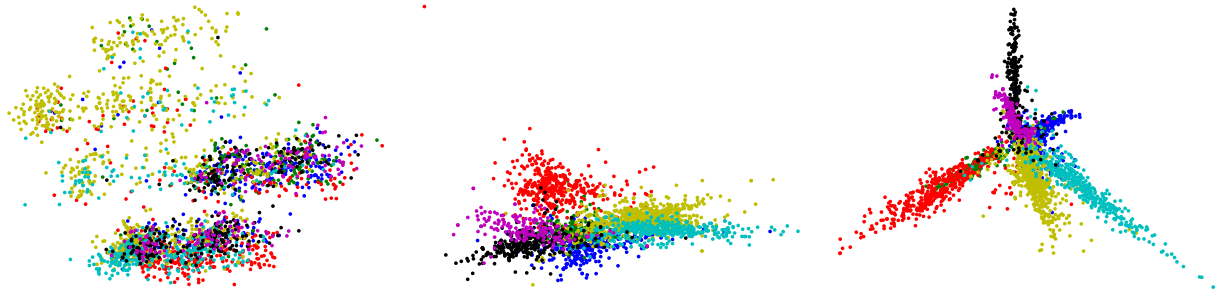
For the comparison between the number of the convolution layer, we set the other hyper-parameters be the same. As illustrated in Table 4, for the transductive setting, we can observe that the classification accuracy of stacking 2 convolution layers is higher than the other two over citation datasets. When $L = 1$, the experimental results are similar to $L = 3$. The reason may be that the receptive field scale is set to 3, even only one convolution layer, the information of the hash aggregating neighbors is sufficient. But also only low-level features have been extracted with one convolution layer, as a result, the performance is not so good. And for the three-layer convolution, in fact, the receptive field and the number of layers are set relative larger, which may cause excessive fusion of node information and ultimately leads to poor classification. For inductive learning, note that we set the receptive filed $S = 2$, and the accuracy of $L = 1$ is significant lower than the other two cases. In the absence of structural information in training, only one layer of convolution can not extract more useful features

When comparing the scale of the receptive fields, the number of convolution layers is 2 for transductive while 3 for inductive. The receptive field size $S = 1$ means that only the information of the node itself is used, and HGC degenerates to an MLP network. Therefore, the performance is not desirable for both transductive and inductive settings. As more information is aggregated, the accuracy of $S = 3$ is generally superior to $S = 2$ for citation datasets. This again verifies the importance of local neighborhood information, which is also an crucial property in traditional convolutional neural networks. Due to the complexity of the calculation and memory constraints, experiments aren't carried out when $S \geq 4$, but we conclude that the larger receptive field may incorporate information from many other categories of nodes, which is not conducive to the final identification.

Table 4: Comparisons on the layer number L and the size of receptive field S.

	L, S	Cora	Citeseer	Pubmed	PPI
Layer Num	$L = 1$	82.3%	72.9%	79.6%	89.2%
	$L = 2$	**85.2%**	**74.3%**	**81.5%**	98.0%
	$L = 3$	83.9%	71.7%	79.5%	**98.6%**
Rcep Size	$S = 1$	60.4%	61.1%	77.1%	54.9%
	$S = 2$	84.4%	73.3%	80.4%	**98.6%**
	$S = 3$	**85.2%**	**74.3%**	**81.5%**	-

Analysis about the number of hash buckets b and the value of regularizer coefficient λ. The degree of the node obeys the long tail distribution, i.e., the degree of most nodes is small. We also demonstrate this in Figure 5, which visualizes the number of nodes in range $(0, 20)$ of degree over three citation datasets. We can found that the number of nodes drops sharply first and then tends to be flat on small values as the degree increases. Obviously, the average degree of citation network is also small according to Table 1. Thus the value of buckets b should be a relative small value. Further, we think a small b helps to eliminate ambiguity and produces more discriminative features through hash-aggregation, However, the features will be excessively refined, over-fitting may occur as the b increases, which will lead to decreased accuracy. We can the see the change of accuracy when b is in range $(1, 10)$ from Figure 7. Classification effect is not good and unstable where b is small or large. And the accuracy is relatively stable and obtain the maximum when b is in range $(5, 8)$ over three citation datasets. Returning to Figure 5, in fact, we can observe that the value of b corresponds to the range of degree, where the degree is relatively large and the number of nodes is also relatively large. Within this range, our HGC model can obtain superior performance. For the value of regularizer coefficient λ, we follow the work [34] to determine the value of regularizer coefficient λ, which is a temporal weight function. At



(a) Original data (b) GCN prediction (c) Ours HGC prediction

Figure 6: Visualization of prediction results on Cora dataset: Original data *vs* GCN *vs* HGC (Ours).

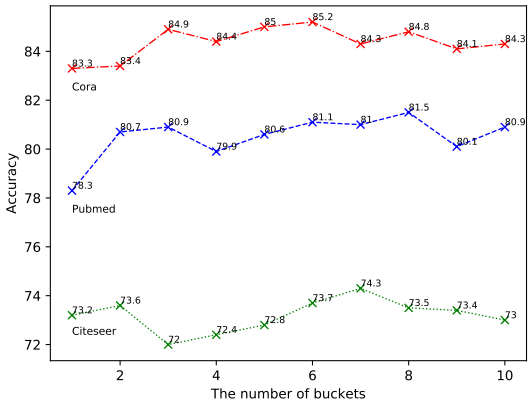


Figure 7: Comparisons on the number of buckets b.

the beginning $\lambda = 0$, then λ gradually increases with the epochs increasing. The specific λ depends on the values of $\mathcal{L}$ and $\mathcal{L}_{Reg}$. However, for inductive learning, testing set and training set have different graph structures, and the structural constraint is less effective and can even be counterproductive, degrading performance. So a small weight is given to the regularizer.

HGC model can be treated as "GCN" when removes the regularizer and hash-projection. "GCN + hashing" is HGC model with the regularizer removed, and "HGC - hashing" means that hash-projection is removed. In order to measure the effect of hash-projection, we conduct two sets of experiments:"GCN" *vs* "GCN + hashing" and "HGC - hashing" *vs* HGC. The corresponding results have been presented in Table 5. We can see that " GCN + hashing " and HGC both have a remarkable gain compared to GCN and "HGC - hashing" on all four datasets. This again adequately proves that hash-projection can preserve discriminative information than conventional averaging/sum aggregation. Meantime, the hash-projection is flexible to be embedded in other models.

Also, Figure 6 shows the visual results on Cora dataset. Figure 6(a) is the original unprocessed data projected into 2D space through principal component analysis. It can be seen that the samples of all classes are mixed together. After training by GCN and our method HGC, the visualization results are shown in Figure 6(b) and Figure 6(c) respectively. We can observe that the classification boundary of HGC is clearer than GCN, the separation distance between different classes is relatively large, and the classification effect is better.

Table 5: The verification of hash-projection.

Method	Cora	Citeseer	Pubmed	PPI
GCN	81.2%	70.6%	78.4%	97.6%
GCN + hashing	82.9%	71.9%	79.2%	98.3%
HGC - hsahing	83.3%	73.2%	78.3%	97.3%
HGC	**85.2%**	**74.3%**	**81.5%**	**98.6%**

5 CONCLUSION

In this paper, we present a simple and effective hashing graph convolution method for node classification task. To our knowledge, we first time introduce hashing into graph convolution. Due to the good locality property, hashing aggregation can better preserve the node discriminative information and reduce the feature confusion caused by constantly aggregating neighborhood information. Another, hash projects the local receptive field of each node into a common-size bucket space, and makes graph convolution analogical to the standard shape-girded convolution. Meantime extensive experiments on both transductive and inductive learning have demonstrated that HGC is superior to the previous work and we achieve state-of-the-art performances in all datasets. In the future, we would like to extend our hash transform into more applications to graph.

REFERENCES

[1] Josh Attenberg, A Dasgupta, J Langford, A Smola, and K Weinberger. 2009. Feature hashing for large scale multitask learning. In *ICML*.
[2] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *NIPS*. 1993–2001.
[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. *ICLR* (2013).
[4] John A Bullinaria and Joseph P Levy. 2007. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods* 39, 3 (2007), 510–526.
[5] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka, and Tom M Mitchell. 2010. Toward an architecture for never-ending language learning. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[6] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rytstxWAW

[7] Fan RK Chung. 1997. *Spectral graph theory*. Number 92. American Mathematical society.

[8] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. 2010. A sparse johnson: Lindenstrauss transform. In *Proceedings of the forty-second ACM symposium on Theory of computing*. ACM, 341–350.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.

[10] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.

[12] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[13] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. 2010. Aggregating local descriptors into a compact image representation. In *CVPR*. IEEE Computer Society, 3304–3311.

[14] Jiatao Jiang, Zhen Cui, Chunyan Xu, and Jian Yang. 2019. Gaussian-Induced Convolution for Graphs. In *Proceedings of the Thirty-Third Conference on Association for the Advancement of Artificial Intelligence (AAAI)*.

[15] William B Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics* 26, 189-206 (1984), 1.

[16] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *ICLR* (2016).

[17] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*. 2177–2185.

[18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[19] Qimai Li, Xiao-Ming Wu, Han Liu, Xiaotong Zhang, and Zhichao Guan. 2019. Label Efficient Semi-Supervised Learning via Graph Filtering. In *Conference on Computer Vision and Pattern Recognition*. https://arxiv.org/abs/1901.09993

[20] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *CVPR*. 5115–5124.

[21] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[23] Jorge Sánchez and Florent Perronnin. 2011. High-dimensional signature compression for large-scale image classification. In *CVPR 2011*. IEEE, 1665–1672.

[24] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, Alex Strehl, and Vishy Vishwanathan. 2009. Hash kernels. In *Artificial intelligence and statistics*. 496–503.

[25] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* 30, 3 (2013), 83–98.

[26] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW (WWW '15)*. 1067–1077. https://doi.org/10.1145/2736277.2741093

[27] Andrea Vedaldi and Andrew Zisserman. 2012. Sparse kernel approximations for efficient classification and detection. In *CVPR*. IEEE, 2320–2327.

[28] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[29] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2012. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 12 (2012), 2393–2406.

[30] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2018. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2018), 769–790.

[31] Bo Wu, Yang Liu, Bo Lang, and Lei Huang. 2017. DGCNN: Disordered Graph Convolutional Neural Network Based on the Gaussian Mixture Model. *arXiv preprint arXiv:1712.03563* (2017).

[32] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536* (2018).

[33] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-supervised Learning with Graph Embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. 40–48.

[34] Chenyi Zhuang and Qiang Ma. 2018. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. WWW, 499–508.

[35] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, 14 (2017), i190–i198.