A Probability Distribution Learning Method for Extracting Key Vertex Pairs in Graph Classification Tasks

Jianming Huang[1,a)] Zhongxi Fang[2,b)] Hiroyuki Kasai[2,c)]

Abstract: Graph classification is a hot topic of machine learning for graph-structured data, and it is also a very potential and valuable research because it has a wide range of applications such as bioinformatics, computer vision, social networks, and so on. However, the difficulty of graph classification is challenging and special, which is different from the ones of normal classification problems. One of the most difficult points of graph classification is that, the number of vertex neighbors in a graph tends to be variable, which makes the number of weights uncertain and ambiguous. In order to overcome these difficulties, we propose a novel method which weights the vertex neighbors based on a weighting function by learning the probability distributions of vertex pairs. The numerical evaluations show that our proposed method outperforms many state-of-the-art methods including some deep learning methods.

1. Introduction

Graph-structured data have been used widely in various fields, such as chemoinformatics, bioinformatics, social networks, and computer vision [1], [2]. Therefore, there also exist many classification tasks for the graph-structured data, such as the toxicity analysis of compounds and recognitions of handwrittings.

However, the graph classification has its difficulties, most of which quite differ from other classfication problems of computer vision and natural language processing. One of the biggest problems is that, unlike the matrix data of a image or the sequence data of a sentence, the number of neighbors of a vertex in a graph is usually variable and uncertain. As shown in Figure 1, this makes it hard to learn the weights of neighbors because we cannot define a size-fixed parameter such like a convolution kernel in the Convolution Nerual Network (CNN), or a window of sequences in the Recurrent Neural Network (RNN). In this case, a continuous function based on features of vertex pairs might help resolve this difficulty, because this continuous weighting function can dynamically compute the weight when inputs are given. In recent work of the Graph Attention Networks (GAT) [4], they propose a Graph Nerual Network (GNN) based on the transformer [5], where they use the following weighting func-

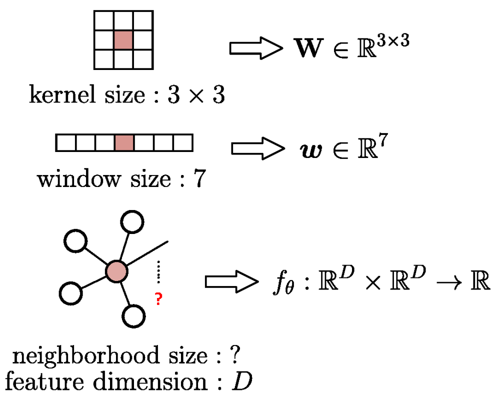
$\mathbf{W} \in \mathbb{R}^{3\times 3}$
kernel size : 3×3
$\boldsymbol{w} \in \mathbb{R}^7$
window size : 7
?
$f_\theta : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$
neighborhood size : ?
feature dimension : D

Fig. 1 A vertex tends to have varible number of neighbors, which makes it hard to learn size-fixed parameters. Therefore, a weighting function based on features of vertex pairs might help.

tion:

$$f_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\exp\left(\mathrm{LeakyReLU}(\boldsymbol{a}[\mathbf{W}\boldsymbol{x}_i || \mathbf{W}\boldsymbol{x}_j])\right)}{\sum_{k\in\mathcal{N}(v_i)} \exp\left(\mathrm{LeakyReLU}(\boldsymbol{a}[\mathbf{W}\boldsymbol{x}_i || \mathbf{W}\boldsymbol{x}_k])\right)},$$

where $\boldsymbol{x}_i, \boldsymbol{x}_j$ denote the feature vector of two vertices, $\boldsymbol{a}, \mathbf{W}$ are the learned parameters, the former is a vector and the later is a matrix. $||$ is the concatenation operation. The value of this function is called the weight of *attention*. The GAT brought great success to the node classification researches. Nevertheless, some questions are still remained: is this equation proper for learning attention? What information is important for learning attention? Will it suffer from the over-fitting problem?

In order to find out what information is advantageous to the attention model, and make an effective attention model

[1] WASEDA University, Graduate School of Fundamental Science and Engineering
[2] WASEDA University, School of Fundamental Science and Engineering, Dept. of Communications and Computer Engineering
[a)] koukenmei@toki.waseda.jp
[b)] fzx@akane.waseda.jp
[c)] hiroyuki.kasai@waseda.jp

for the graph classification, we propose a novel method which learns the weighting function by applying the Gaussian Mixture Model (GMM) to the probability distribution of vertex pairs. We conduct evaluations on several real world benchmark datasets and found that our proposed method outperforms many state-of-the-art methods in graph classification tasks.

2. Related Work

For graph classification tasks, Graph Kernel (GK) methods were the mainstream methods and it has been used widely for several decades before Graph Neural Networks (GNN) came out. Theoretically, GKs are kernel functions which could compute similarity for graph pairs. Most of GKs are based on the isomorphism and structural similarity of graph-structured data. In earlier researches, GKs have shown its effectiveness for graph classification tasks with machine learning algorithms including the Support Vector Machine (SVM). A famous work of GK is the Weisfeiler–Lehman (WL) Graph Kernel [7], which brought great success in this domain and it is still inspiring many state-of-the-art reseaches now. In WL graph kernel, They proposed a similarity metric based on the Weisfeiler–Lehman test and implement it as a general framework. To improve the similarity metric of the WL graph kernel and make it more robust for the decomposite graph structure, related work [6] brought an optimal assignment kernel variant from the WL graph kernel, which is called Weisfeiler–Lehman Optimal Assignment (WL-OA) graph kernel. It is based on an optimal bijection between substuctures of graph pairs, which performs better and more robust than the original WL graph kernel. Similar to the WL-OA graph kernel, there is a research [8] proposing a Wasserstein-based Weisfeiler–Lehman (WWL) Graph Kernel, which maps a node embedding based on its neighborhood pattern to a novel feature space. In this feature space, they compute distance of graphs based on the Wasserstein distance of two point clouds where a single point denotes a node of graphs.

In recent years, as the large-scale datasets become more important, the shortcomings of GKs become more obvious: the high cost of both the computational complexity and the memory. Thus, GNNs become hot topics in graph domain including the graph classification, the node prediction and the link prediction. A representative method of GNN in recent years is the Graph Convolution Network (GCN) [9], which is inspired by the Convolution Neural Network (CNN) in computer vision domain and applies the same way to deal with graph-structured data. Another representative GNN is the Graph Isomorphism Network (GIN) [10]. The GIN is inspired by the WL graph kernel which specifically examines graph isomorphism. They prove that the SUM aggregation generates better aggregation than other schemes including MEAN and MAX, in message passing process of graphs. They apply this scheme in GIN, which make it more powerful than other GNNs in graph classification tasks.

3. Probability Distribution Learning for Binary Classification

In this section, to make it easy to understand, we first propose a probability distribution learning scheme of vertex pairs under a simple task: the binay classification. We will then expand it for the multi-class classfication in the next section.

3.1 Step 1: Sample the vertex pairs and map them to a discrete distribution

Firstly, as shown in Figure 2, before the sampling, we should compute the feature vector of vertex pairwise which will form the discrete distribution. Let $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$ be the feature of two connected vertices and let $f_{\text{pair}} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ be the function aggregating features of these two vertices, the aggregated feature of vertex pair is

$$f_{\text{pair}}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1 + \boldsymbol{x}_2.$$

After the aggregation, we can obtain a d-dimensional vector for each connected vertex pairs, which can be mapped to a d-dimensional Euclidean metric space.

For the process of sampling, let us consider a binary classification task, we first divide the graphs into 2 subsets corresponding to their class labels. Next, all connected vertex pairs of graphs in the subset of i-th class label will be mapped to a distribution $\mathcal{D}_i$. Therefore, we can get 2 discrete distrbutions $\mathcal{D}_1, \mathcal{D}_2$.

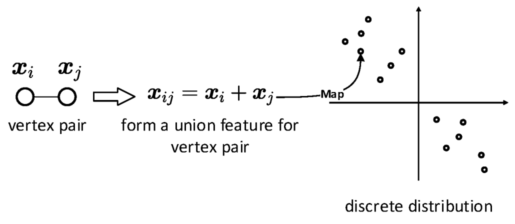
$\boldsymbol{x}_i$ $\boldsymbol{x}_j$
$\boldsymbol{x}_{ij} = \boldsymbol{x}_i + \boldsymbol{x}_j$
Map
vertex pair
form a union feature for vertex pair
discrete distribution

Fig. 2 Overview of step 1, where connected vertex pairs will be mapped to a discrete distribution.

3.2 Step 2: Use the GMM learn the probability distribution

Use the discrete distribution directly will bring great difficulties for computation, which may lead to high computational costs and memory costs because of frequent point comparison. Therefore, we use distribution learning methods such as GMM to learn continuous probability distributions of vertex pairs as shown in Figure 3.

3.3 Step 3: Compute a combined distribution

Through analyzing the obtained continuous distributions, we found that there exist independent parts in these distributions which are independent to the class labels. As shown in Figure 4, the red part does not change too much w.r.t class label 1 and class label 2. These part should be considered as features less valuable for classification. In order to filter out these independent parts, we apply a simple way to combine

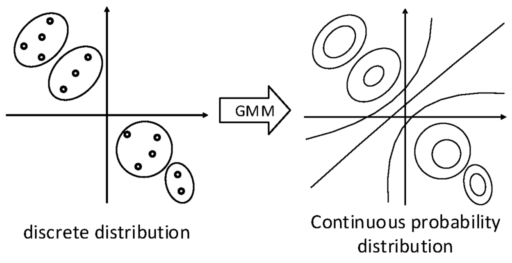

Fig. 3 Overview of step 2, where we use the GMM to learn a continuous probability distribution.

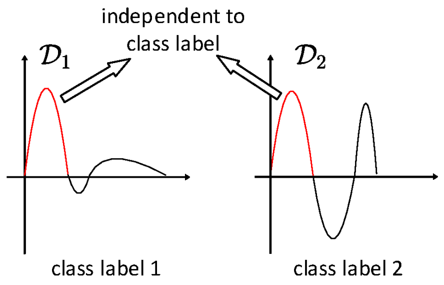

Fig. 4 Illustration of the independent parts in distributions.

two distribution as:

$$\mathcal{D}' = |\mathcal{D}_1 - \mathcal{D}_2|,$$

where $\mathcal{D}_1, \mathcal{D}_2$ are two distributions and $\mathcal{D}'$ is their combined distribution.

3.4 Step 4: Update each vertex feature

Similar to most graph classification methods, we apply a message-passing-based framework to update and aggregate vertex features in graphs. In this step, feature of each vertex will be updated by aggregating its and its neighbors' features. Let $\boldsymbol{x}_i \in \mathbb{R}^d$ be the feature vector of i-th vertex in a graph, then it will be updated as

$$\boldsymbol{x}'_i = \sum_{j \in \mathcal{N}(v_i)} f_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j) \cdot \boldsymbol{x}_j,$$

where $f_\theta : \mathbb{R}^D \times \mathbb{R}^\theta \to \mathbb{R}$ is:

$$f_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\exp\left(\mathcal{D}'(\boldsymbol{x}_i + \boldsymbol{x}_j)\right)}{\sum_{k \in \mathcal{N}(v_i)} \exp\left(\mathcal{D}'(\boldsymbol{x}_i + \boldsymbol{x}_k)\right)},$$

where $\mathcal{N}(v_i)$ denotes the set of neighbors of i-th vertex.

The step 1 to step 4 will repeat H iterations for updating. If the process of updating ends, all vertex features will be inputted into a global mean pooling layer to get a graph representation, which will be used for classification.

4. Class-wise Feature Updating for Multi-class Classification

So far, the feature updating process w.r.t a single class-label is described in the last section, which can be directly used in the binary classification task. However, there are many classification tasks that have more than 2 class labels. In order to expand our method so that it can work in the multi-class classification task, we conduct a separate-and-concatenate framework which is as shown in the Fig. 5.

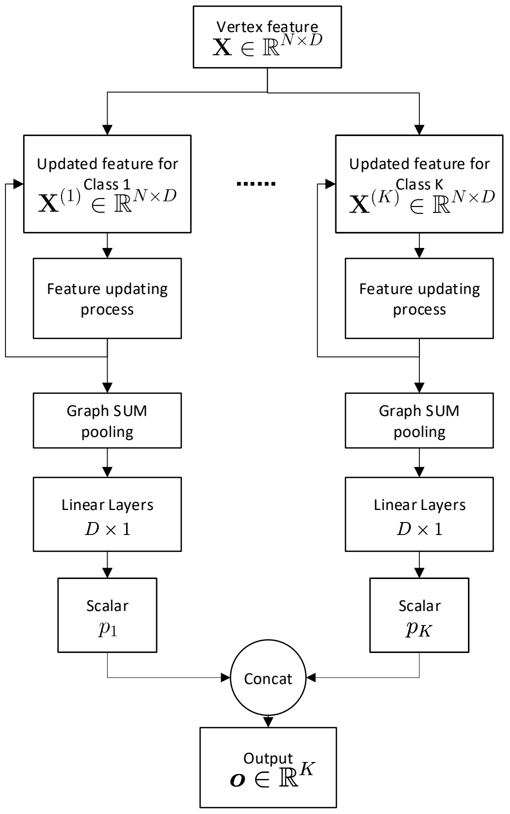
Vertex feature $\mathbf{X} \in \mathbb{R}^{N\times D}$
Updated feature for Class 1 $\mathbf{X}^{(1)} \in \mathbb{R}^{N\times D}$
......
Updated feature for Class K $\mathbf{X}^{(K)} \in \mathbb{R}^{N\times D}$
Feature updating process
Feature updating process
Graph SUM pooling
Graph SUM pooling
Linear Layers $D \times 1$
Linear Layers $D \times 1$
Scalar p_1
Scalar p_K
Concat
Output $\boldsymbol{o} \in \mathbb{R}^K$

Fig. 5 Illustration of the framework for the multi-class classification task, where the vertex features of a single graph will be updated separately w.r.t each class label and finally they will be concatenated into an ouput vector.

Let G be the graph that we are going to transform and classify, $\mathbf{X} \in \mathbb{R}^{N\times D}$ denotes its vertex features, which expresses that it has N vertices and each vertex is embedded into a D-dimensional vector. Assume that we have K class labels. In this case, we first copy the vertex features and repeat them into K channels, which are denoted as $\mathbf{X}^{(1)}...\mathbf{X}^{(K)}$. The vertex features in K channels will be updated separately and independently for a proper number of iterations, w.r.t the class label that they belong to. This means that, in the probability distribution learning step of the i-th channel, graphs will be divided into two subsets: belonging to class label i and not belonging to class label i. Then the probability distribution is learned by following the steps described in the last section. After we finish the feature updating process, we will conduct a graph SUM pooling on the vertex features, where the vertex features will be added together to form a new embedding for the graph G. Then we will apply several linear layers to $\mathbf{X}^{(1)}...\mathbf{X}^{(K)}$ separately, which take an input of D dimension and give an output of 1 dimension. Then we can obtain K scalars $p_1..p_K$, each of them denotes the probability of belong to their class labels. Finally, these scalars will be concatenated together to form an output $\boldsymbol{o} \in \mathbb{R}^K$, which will be inputted into a log-softmax classifier.

5. Numerical Evaluation

We conduct evaluation experiments on several widely-

Table 1 Average classification accuracy on graph datasets

METHOD	MUTAG	PTC-MR	PROTEINS	COX2
WL $H = 4$	85.61±8.02	62.51±4.11	74.24±3.75	81.36±3.21
WWL $H = 4$	**85.90±7.39**	**65.31±7.06**	74.13±3.47	81.75±3.71
WL-OA $H = 4$	82.72±7.09	63.45±8.63	73.83±3.61	80.47±4.44
GIN	**88.59±6.89**	64.76±7.67	73.72±4.27	**82.59±4.42**
unweighted	74.03±8.79	60.16±9.03	**76.82±3.82**	78.15±0.80
weighted	80.87±10.53	**65.42±7.40**	**75.74±4.31**	**81.99±4.27**

used real world benchmark datasets, which are the MUATG [11], the PTC-MR [12], the PROTEINS [13] and the COX2 [14] datasets, where we use the Support Vector Machine (SVM) to do classification. For each dataset, I conduct one time of 10-fold nested cross validation to get the average accuracy and standard deviation. For the parameter setting, we use the GMM with 100 clusters and update vertex features for $H = 4$ iterations.

For the comparing methods, we choose 4 state-of-the-art works, which are the Weisfeiler-Leman graph kernel (WL) [7], the Wasserstein Weisfeiler-Leman graph kernel (WWL) [8], the Weisfeiler-Leman graph kernel with Optimal Assignment (WL-OA) [6] and the Graph Isomorphism Nerual Network (GIN) [10]. In order to show the difference between weighted strategy and unweighted strategy, we also add an unweighted variant of our proposed method, where we update vertex feature by the following equation:

$$\boldsymbol{x}'_i = \frac{1}{|\mathcal{N}(v_i)|} \sum_{j \in \mathcal{N}(v_i)} \boldsymbol{x}_j.$$

The experimental results are shown in Table 5, where the top-2 are in bold typeface. From the results we can see that our proposed method outperforms all comparing methods in the PTC-MR, the BZR and the COX2 datasets. For the MUTAG dataset, although our method is not the best, it obtain a nice accuracy which is not far from other methods.

6. Conclusion

In this article, we propose graph classification method based on a weighting function learned by applying the GMM to the probabiliry distributions of vertex pairs. Although the better performance comparing to other methods, our proposed method still has some difficulties: It still has high computational costs for the GMM learning, which makes it hard to be used in large-scale datasets. For improvement, we plan to set the follows as our future work: 1) Apply sampling methods such as the Gibbs sampling to reduce computational costs; 2) Use deep distribution learning methods to learn continuous distributions.

References

[1] Vishwanathan, S.V.N., Schraudolph N.N.. et al.: Graph kernels, *The Journal of Machine Learning Research*, Vol.11, pp.1201–1242, 2010.

[2] Kriege, N.M. and Johansson F.D. et al.: A survey on graph kernels, *Applied Network Science*, Vol.5, No.1, pp.1–42, 2020.

[3] Gary, M.R. and Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness, *New York* (1979).

[4] Veličković, Petar and Cucurull, Guillem et al; Graph attention networks, *International Conference on Learning Representations.*

[5] Vaswani, Ashish and Shazeer, Noam et al.: Attention is all you need, *Advances in Neural Information Processing Systems.*

[6] Kriege, Nils M and Giscard, Pierre-Louis et al.: On valid optimal assignment kernels and applications to graph classification, *Advances in Neural Information Processing Systems*

[7] Shervashidze, N. and Schweitzer, P. et al.: Weisfeiler-Lehman Graph Kernels, *The Journal of Machine Learning Research*, Vol.12, pp.2539–2561, 2011.

[8] Togninalli, M. and Ghisu, E. et al.: Wasserstein weisfeiler-lehman graph kernels, *Advances in Neural Information Processing Systems* pp.6439–6449, 2019.

[9] Kipf, Thomas N and Welling, Max: Semi-supervised classification with graph convolutional networks, *International Conference on Learning Representations.*

[10] Xu, Keyulu and Hu, Weihua et al.: How powerful are graph neural networks?, *International Conference on Learning Representations.*

[11] Debnath, Asim Kumar and Lopez de Compadre, Rosa L et al.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity, *Journal of Medicinal Chemistry*, Vol.34, No.2, pp.786–797 (1991).

[12] Helma, Christoph and King, Ross D. et al.: The predictive toxicology challenge 2000–2001, *Bioinformatics*, Vol.17, No.1, pp.107–108 (2001)

[13] Borgwardt, Karsten M and Ong, Cheng Soon et al.: Protein function prediction via graph kernels, *Bioinformatics*, Vol.21, No.suppl_1, pp.i47–i56 (2005).

[14] Sutherland, Jeffrey J and O'brien, Lee A et al.: Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships, *Journal of Chemical Information and Computer Sciences*, Vol.43, No.6, pp.1906–1915 (2003).