An End-to-End Deep Learning Architecture for Graph Classification
Final Report

Jie Tang, Menghong Han, Xiaofan Sun

1 Background and Introduction

The authors of this paper Muhan Zhang et al. proposed a new architecture for neural network on graph classification.Neural networks are typically designed to deal with data in tensor forms. The past few years have seen the growing prevalence of neural networks on application domains such as image classification (Alex, Sutskever, and Hinton 2012), natural language processing (Mikolov et al. 2013), reinforcement learning (Mnih et al. 2013), and time series analysis (Cui, Chen, and Chen 2016).

The connection structure between layers makes neural networks suitable for processing signals in tensor forms where the tensor elements are arranged in a meaningful order. This fixed input order is a cornerstone for neural networks to extract higher-level features.

For example, if we randomly shuffle the pixels of an image, then state-of-the-art convolutional neural networks (CNN) fail to recognize it as an eagle.

2 Challenge

This paper aims to develop neural networks that read the graphs directly and learn a classification function. There are two main challenges:

1) how to extract useful features characterizing the rich information encoded in a graph for classification purpose.

2) how to sequentially read a graph in a meaningful and consistent order.

3 Problem Definition

Given a dataset containing graphs in the form of (G, y) where G is a graph and y is its class, to address the first challenge, he designs a localized graph convolution model and shows its connection with two graph kernels. To address the second challenge, he designs a novel SortPooling layer, which is also the highlight and a novel improvement of this method proposed in the paper. The SortPooling layer

can sort graph vertices in a consistent order so that traditional neural networks can be trained on the graphs.

4 Model Explained

4.1 Graph Convolution Layers

After we input a graph, it goes through the first layer of the Graph Convolution Layers. The layer tries to extract local substructure information by this formula,

$$\mathbf{Z} = f(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{X}\mathbf{W})$$

where A hat is adjacency matrix of the graph with added self-loops, D hat is its diagonal degree, W is a matrix of trainable graph convolution parameters, f is a nonlinear activation function, and Z ∈ R n×c is the output activation matrix. XW maps the c feature channels to c ′ channels in the next layer. A hat *Y (Y= XW) propagates node information to neighboring vertices and the node itself. D hat is use d to normalize each row, and after appling a pointwise nonlinear activation function f we get the out put Z. And then the authors applied multiple layers to extract more information by this formula,

$$\mathbf{Z}^{t+1} = f(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{Z}^{t}\mathbf{W}^{t})$$

And then they concatenate the output Z_t , t = 1,…..h horizontally to form a concatenated output, written as $\mathbf{Z}^{1:h} := [\mathbf{Z}^1, \ldots, \mathbf{Z}^h]$ each row can be regarded as a "feature descriptor" of a vertex, encoding its multi-scale local substructure information. Let's see an example to have a clear understand of the shape of the input and output.

	Input	**1st layer**	**2nd layer**	**3rd layer (Final)**	**Output**
Shape	n*15	Z1 (n*5)	Z2 (n*3)	Z3 (n*2)	n*(5+2+3)

Table1. The shape of the output of the Graph Convolution Layers

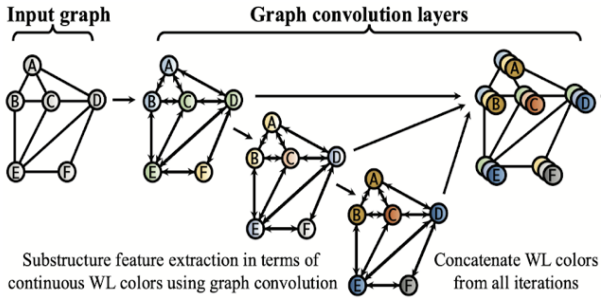

Figure1. Graph Convolution Layers

In addition, the Graph Convolution Layers was inspired by Weisfeiler-Lehman Graph Kernel, which is widely used to check the isomorphism of graphs. The basic idea of WL is to concatenate the color of a vertex with the colors of its neighbors at 1 jump thus concatenate the signature WL of the vertex (b), then to sort the signature strings lexicographically to attribute new colors (c). Vertices with the same signature are given the same new color (d). The procedure is iterated until the colors converge or reach a maximum iteration h. In the end, vertices with the same converged color thus share the same structural role in the graph and are no longer distinguished. If two graphs are isomorphic, they will have the same set of WL colors at each iteration. The core of the WL subtree uses this idea to measure the similarity between two graphs G and G': after iteration, the similarity is determined by the calculation of the kernel function (e).

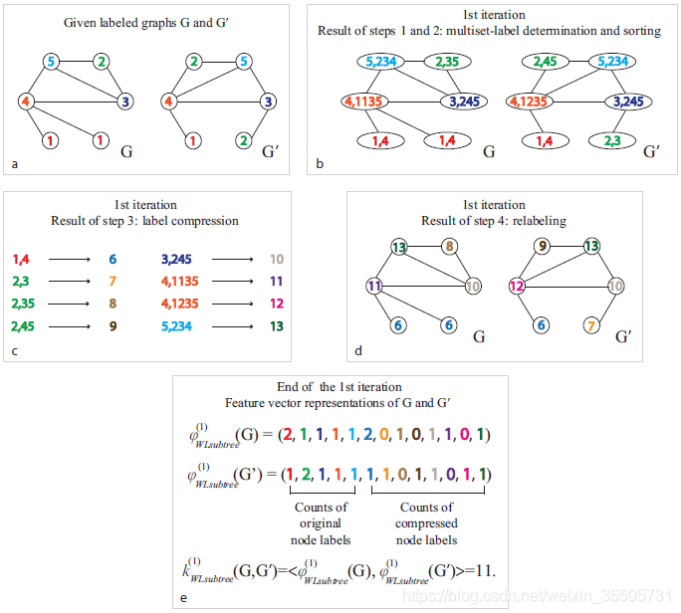

Figure2. WL Graph Kernel

The connection between WL and the Graph Convolution Layers can be seen if we rewrote the formula like this:

$$\mathbf{Z}_i = f([\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}]_i\mathbf{Y}) = f(\tilde{\mathbf{D}}_{ii}^{-1}(\mathbf{Y}_i + \sum_{j\in\Gamma(i)}\mathbf{Y}_j)).$$

The authors view Y which is feature matrix as a continuous color of vertex i, they regard Graph Convolution Layers is a "soft" version of the WL.

4.2 The SortPooling Layers

After we had the features of vertices, traditionally we will summarize the features. While SortPooling can classifie them in a consistent order, it uses structural role of graphs, the continuous WL colors Zt, t = 1, . . . , h to sort vertices.

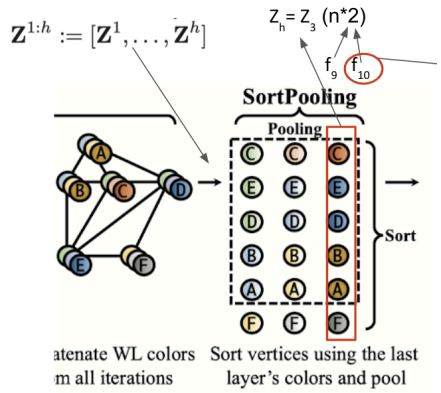

Figure3. The SortPooling Layers

They first sorting vertices using the last channel of Zh in a descending order, finally we got the output $\mathbf{Z}^{sp}$ of size $k \times \sum_1^h c_t$, where k is a user-defined integer so that m% of graphs have nodes more than this integer k. In order to unify the size, they deleted the last n − k rows if n > k, or added k − n zero rows if n < k. The dimension of the output is n*10 in our example.

4.3 The Remaining Layers

Before they feed the output of the SortPooling Layers they reshaped the out into a row-wise tensor so that the traditional CNN can read the vertices directly.

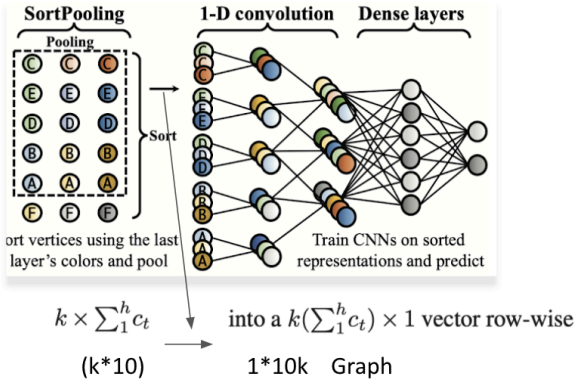

Figure4. The Remaining Layers

After reshaping, they added a 1-D convolutional layer, a MaxPooling layers, a 1-D convolutional layer and a dense layer.

5 Experiments

5.1 Reimplementation result

Using the Bioinformatics datasets: MUTAG, PTC, NCI1, PROTEINS, D&D; social network datasets: COLLAB, IMDB-B, IMDB-M, we reimplemented the experiments

	MUTAG	PTC	NCI1	PROTEINS	D&D	COLLAB	IMDB-B	IMDB-M
Paper Result	**85.83**±1.66	58.59±2.47	**74.44**±0.47	**75.54**±0.94	79.37±0.94	**73.76**±0.49	**70.03**±0.86	47.83±0.85
Reimplementation	83.89±**0.08**	**58.82±0.08**	73.34±**0.02**	74.20±**0.48**	**81.20±0.53**	72.97±0.96	68.7±**0.03**	**48.2±0.37**

Table2. The results of the paper and our implementation

The reimplementation results are almost the same with the paper results. Results of PTC, D&D and IMDB-M datasets are slightly better than paper did, and the rest of datasets perform slightly worse than paper results.

Dataset	MUTAG	PTC	NCI1	PROTEINS	D&D
Nodes (max)	28	109	111	620	5748
Nodes (avg.)	17.93	25.56	29.87	39.06	284.32
Graphs	188	344	4110	1113	1178
DGCNN	**85.83±1.66**	58.59±2.47	74.44±0.47	**75.54±0.94**	**79.37±0.94**
GK	81.39±1.74	55.65±0.46	62.49±0.27	71.39±0.31	74.38±0.69
RW	79.17±2.07	55.91±0.32	>3 days	59.57±0.09	>3 days
PK	76.00±2.69	**59.50±2.44**	82.54±0.47	73.68±0.68	78.25±0.51
WL	84.11±1.91	57.97±2.49	**84.46±0.45**	74.68±0.49	78.34±0.62

Table3. The comparison between DGCNN and other graph kernels in the paper

When we compared the results with other graph kernels, we can see that although a single structure was used for all datasets, DGCNN achieved highly competitive results with the compared graph kernels, including achieving the highest accuracies on MUTAG, PRO- TEINS, and D&D. Compared to WL, DGCNN has higher accuracies on all datasets except for NCI1, indicating that DGCNN is able to utilize node and structure information more effectively.

5.2 Our Improvement Ideas

After reading the codes and comparing with other algorithms, we came up with four ideas of improving the performance of DGCNN:

1) Using 2-hop neighbors or BFS and DFS methods to extract vertices' local substructure features in graph convolution layers
   The paper uses 1-hop neighbor to extract localized features in the first layer, so we are thinking of improving the accuracy by adding one hop neighbor feature extraction or using BFS/DFS to get richer information from neighbors.
2) Deeper CNN or 2-D convolution
   In the last layer of traditional convolutional and dense layer, the authors used 1-D convolution. So we came up with the idea of adding another layer to see if it can performs better.
3) SortingPooling based on last m channels giving higher weights to later channels
   In this paper, the SortPooling order is based on the last channel: The vertex order based on Zh is calculated by first sorting vertices using the last channel of Zh in a descending order. If two vertices have the same value in the last channel, the tie is broken by comparing their values in the second to last channel, and so on. We propose to use the last m channels and give the later channels higher weights, so that it can also consider the previous channels, hopefully reach a better performance and cut the procedure of comparing the values until the ties are broken.
4) Hyperparameter tuning on k
   This paper doesn't involve the model selection procedure and we conducted experiments on different k value to see if it can reach a higher accuracy.

5.3 Comparison between Paper and Our Modeling Process

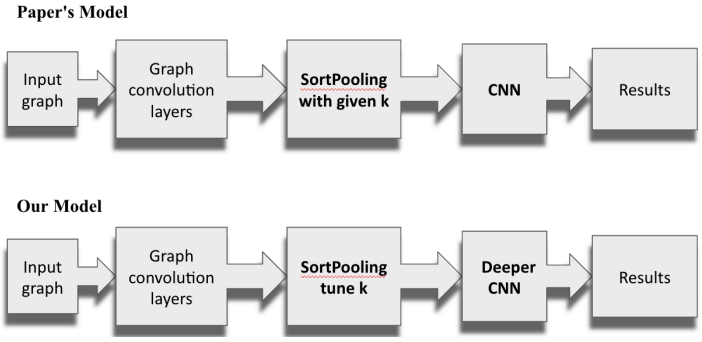

Figure5. Paper's model vs Our model

The difference lies in the k value settings in SortPooling layer and the Deeper CNN model. Here is comparison of results running on the same datasets

Bioinformatics datasets: MUTAG, PTC, NCI1, PROTEINS, D&D; social network datasets: COLLAB, IMDB-B, IMDB-M.

	MUTAG	PTC	NCI1	PROTEINS	D&D	COLLAB	IMDB-B	IMDB-M
Paper Result	**85.83**±1.66	58.59±2.47	**74.44**±0.47	**75.54**±0.94	79.37±0.94	**73.76±0.49**	**70.03**±0.86	**47.83**±0.85
Our Model	82.49±**0.07**	**58.90±0.04**	73.34±**0.02**	74.20±**0.48**	**80.70±0.43**	72.97±0.96	67.65±**0.03**	45.2±**0.47**
Best K	0.6	**0.55**	0.6	0.6	**0.65**	0.9	0.9	0.9

Table4. The comparison between paper's result and ours'

Our model did not thoroughly beat the paper, except on dataset PTC and D&D, which performed slightly better than the paper result. But after experimenting on different k values, we figured out that 0.55 or 0.65 perform better than 0.6, so the tuning on k value is a right choice to improve the algorithm.
Also, our adjusted model has a more stable performance than the paper. The standard variation is much smaller than the paper result.

6 Comments

6.1 Advantages

1) Standard deviations of experiment results are included, which are rarely seen in other paper results
2) Proposed a novel end-to-end deep learning architecture for graph classification. It directly accepts graphs as input without the need of any preprocessing
3) Proposed a novel spatial graph convolution layer to extract multi-scale vertex features and draw analogies with popular graph kernels to explain why it works
4) Proposed a novel SortPooling layer to sort the vertex features instead of summing them up, which can keep much more information and allows us to learn from the global graph topology

6.2 Disadvantages

5) A visualization on experiment results could be more intuitive and easier to compare with other methods
6) No detailed codes of model selection
7) Unclear explanations on connections with WL kernel