DEEP GEOMETRICAL GRAPH CLASSIFICATION WITH DYNAMIC POOLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Most of the existing Graph Neural Networks (GNNs) are the mere extension of the Convolutional Neural Networks (CNNs) to graphs. Generally, they consist of several steps of message passing between the nodes followed by a global indiscriminate feature pooling function. However, most of the times the nodes are unlabeled or their labels (or the given feature vectors of the nodes) provide no information about the similarity between the nodes and the locations of the nodes in the graph. Accordingly, message passing may not propagate helpful information throughout the graph. We show that this conventional approach can fail to learn to solve even simple graph classification tasks. We alleviate this serious shortcoming of the GNNs by making them a two step method where in the second step, the message passing block is given the continuous features obtained by the embedding algorithm in the first step. The GNN learns to solve the given task by inferring the topological structure of the graph encoded in the spatial distribution of the embedded vectors. The second challenge we address in this paper is designing a pooling algorithm applicable to graphs. We turn the problem of graph down-sampling into a column sampling problem, i.e., the sampling algorithm samples a subset of the nodes whose feature vectors preserve the spatial distribution of all the feature vectors. We apply the proposed approach to several established benchmark datasets and it is shown that the proposed geometrical approach strongly improves the state-of-the-art for several data-sets.

1 INTRODUCTION

Many of the modern data/signals are naturally represented by graphs (Bronstein et al., 2017; Dadaneh & Qian, 2016; Cook & Holder, 2006; Qi et al., 2017) and it is of great interest to design data analysis algorithms which can work directly with graphs. Due to the remarkable success of the deep learning based methods in many machine learning applications, it is an important research problem to design new architectures which can make the deep networks able to work with graphs. The adjacency of the graph shows the local connectives of the nodes. Thus, it is straightforward to extend the local feature aggregation in the Convolutional Neural Networks (CNNs) to the Graph Neural Networks (GNNs) (Simonovsky & Komodakis, 2017; Zhang et al., 2018; Atwood & Towsley, 2016; Niepert et al., 2016; Bruna et al., 2013; Fey et al., 2018). The local feature aggregation in the graphs is equivalent to message passing between the nodes (Zhang et al., 2018; Gilmer et al., 2017). However, if the graph is not labeled (the nodes/edges do not have any attribute) or the feature vectors of the nodes carry no information about the similarities between the nodes or their structural role, the message passing may not propagate informative features throughout the graph. Thus, the GNN can fail to infer the topological structure of the graphs. Another limitation of the current GNN architectures is that they are mostly unable to do the hierarchical feature learning as in the CNNs (Krizhevsky et al., 2012; He et al., 2016). The main reason is that graphs lack the tensor representation and it is hard to measure how accurate a sub-set of the nodes represent the topological structure of a graph. In this paper, we address these two shortcomings of the GNNs.

In the proposed approach, we provide the GNN a spatial representation of the graph in the embedding space in which the topological structure of the graph is encoded. In the proposed approach, the network is aware of the similarities/differences between the nodes and also the locations of the nodes in the graph. Accordingly, the local feature aggregation can propagate informative messages throughout the graph. In addition, we propose a feature sampling approach to perform graph pool-

ing. The proposed node sampling approach measures the similarities between the nodes in the spatial domain and merges the closest pairs of nodes. The main contributions of the proposed approach can be summarized as follows:

- The proposed approach enhances the ability of the GNNs in inferring the topological structure of the graphs.
- A new graph pooling method is proposed which can be implemented in any GNN. The proposed pooling method preserves the spatial distribution of the continuous representation of the graph by merging the closet (close in the spatial domain) pairs of nodes.
- To the best of our knowledge, the proposed approach advances the state-of-the-art results for 5 established benchmark data-sets.

1.1 NOTATION

We use bold-face upper-case letters to denote matrices and bold-face lower-case letters to denote vectors. Given a vector $\mathbf{x}$, $\|\mathbf{x}\|_p$ denotes its ℓ_p Euclidean norm, and $\mathbf{x}(i)$ denotes its ith element. Given a set $\mathcal{A}$, $\mathcal{A}[i]$ denotes the ith element of the set. Given a matrix $\mathbf{X}$, $\mathbf{x}_i$ denotes the ith row of $\mathbf{X}$ and $\mathbf{X}(i,j) = \mathbf{x}_i(j)$. A graph with n nodes is represented by two matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{X} \in \mathbb{R}^{n\times d_1}$, where $\mathbf{A}$ is the adjacency matrix, $\mathbf{X}$ is the matrix of feature vectors of the nodes, and d_1 is the dimension of the attributes of the nodes. The matrix of feature vectors at the input of the lth layer of the network is denoted by $\mathbf{X}^l \in \mathbb{R}^{n^l\times d^l}$, where n^l is the size of the graph at the output of the $(l-1)^{\text{th}}$ layer of the network and d^l is the length of the feature vectors at the output of the $(l-1)^{\text{th}}$ layer of the network ($\mathbf{X}^0 = \mathbf{X}$ and $n^0 = n$). The set $\mathcal{I}_i$ contains the indices of the neighbouring nodes of the ith node. We also include i in $\mathcal{I}_i$. The operation $\mathbf{A} \Leftarrow \mathbf{B}$ means that the content of $\mathbf{A}$ is set equal to the content of $\mathbf{B}$. Given a node in a graph, the neighbour set of the given node is a set composed of the neighbouring nodes of the given node. The function $\lfloor y \rfloor$ returns the closet integer which is smaller than or equal to the real number y.

2 RELATED WORK

The proposed approach is a new deep learning based method for the graph classification task. In the proposed method, the deep network is given the geometrical representation of the graph which is obtained by a graph embedding algorithm. Therefore, in this section we review some of the related research works in graph classification and graph embedding.

Graph Embedding: A graph embedding method aims at finding a continuous feature vector representation of the graph such that the topological structure of the graph is encoded in the spatial distribution of the feature vectors. The nodes which are closely connected on graph or they share similar structural role are mapped to nearby points. The graph embedding methods such as DeepWlak (Perozzi et al., 2014) and Node2Vec (Grover & Leskovec, 2016) generalize the recent advances in world embedding to graphs. For instance, DeepWalk uses the local information obtained from the random walks to learn the nodes representations by treating the random walks as sentences. We refer the reader to (Ivanov & Burnaev, 2018; Perozzi et al., 2014; Grover & Leskovec, 2016) and the references therein for a more comprehensive review of graph embedding.

Kernel Based Graph Classification Methods: Graph kernels are graph analysis tools which are able to measure the similarity between two graphs. They make the kernel machines such as SVM able to work directly on graphs. The primary idea was proposed in Haussler (1999) which decomposes a graph into small substructures and computes the similarity between two graphs by adding up the pair-wise similarities between these components. The main difference between the graph kernels is in their choice of the substructures. The substructures include walks (Vishwanathan et al., 2010), sub-graphs (Kriege & Mutzel, 2012), paths (Borgwardt & Kriegel, 2005), and sub-trees (Shervashidze et al., 2011). Despite the success of graph kernels, they have two major drawbacks. First, the graph kernels are computationally expensive. These methods need to fill out the kernel matrix by computing the similarity between every two graphs in the training data-set. Thus, the complexity of training them scales with the square of the size of the training data-set. In addition, computing each element of the kernel matrix can be computationally expensive. For instance, the computation complexity of the shortest path graph kernel scales with the square of the number of nodes. This

high computation complexity make the algorithm inapplicable to large graphs. In Section 6.2, we compare the performance of the proposed method with some of the graph kernels and it is shown that although the computation complexity of the proposed approach can be linear with the number of nodes and the complexity of the training of the proposed method is linear with the size of the training data, it strongly outperforms them on most of the data-sets. The second problem with graph kernels is that the features that they use for classification is independent from the data-set. In contrary to the deep learning based methods, the extracted features are not data driven.

Graph Neural Networks: In recent years, there has been a surge of interest in developing deep network architectures which can work with graphs. The main trend is to adopt the structure of the convolutional networks. In contrary to images, we do not have any meaningful order of the neighbouring nodes in graphs. Thus, we can not assign a specific weight to each neighbouring node of a given node. Accordingly, the feature vector of a given node is uniformly aggregated with the feature vectors of its neighbouring nodes. Specifically, the matrix of feature vectors $\mathbf{X}$ is updated as $\mathbf{X} \Leftarrow f\left(\mathbf{AXW}\right)$, where $\mathbf{W}$ is the weights matrix assigned to all the nodes and $f(\cdot)$ is the element-wise non-linear function. If this operation is repeated k-times, a given node receives "messages" from all the nodes with distance up to k steps away (Gilmer et al., 2017).
In addition to the lack of tensor representation in graphs, it is not straightforward to perform pooling on graphs. Thus, the standard approach in GNNs is to perform the local feature aggregation (message passing) for few times and subsequently aggregate (mostly using element-wise mean function or element-wise max function) all the feature vectors at once. In order to avoid the aggregation over the whole graph and to "keep more vertex information", the recent architecture presented in Zhang et al. (2018) builds a sequence of a set of sampled nodes and applies a one dimensional CNN to the sequence of the nodes. However, it is evident that a one dimensional space do not have the capacity to represent the structure of a graph. For instance, we lose all the spatial information in an image if we sort the pixels (or their feature vectors) in a 1-dimensional array. In Ying et al. (2018), a soft graph pooling method was proposed but with the pooling method proposed in Ying et al. (2018), the sparsity of $\mathbf{A}$ is lost and this makes message passing computationally expensive for large graphs. In (Defferrard et al., 2016; Fey et al., 2018), the graph clustering algorithms were combined with the GNN to perform the graph down-sizing. In this paper, we do not assume that the given graph is clustered and the graph pooling is performed dynamically with respect to the distribution of the learned feature vectors.

3 MOTIVATIONS

Consider a 2-dimensional image. This image is represented using a 3-dimensional tensor. A given pixel in the image is surrounded by 8 pixels. Each neighbouring pixel has a distinct relative position with respect to the given pixel. This spatial order makes the CNN able to aggregate the local feature vectors using different weight matrices $\{\mathbf{W}_i\}_{i=0}^{8}$. Thus, if $\mathbf{x}_0$ represents the feature vector of the center pixel and $\{\mathbf{x}_k\}_{i=1}^{8}$ represent the feature vectors of the surrounding pixels, the feature vector of the center pixel is updated as $\mathbf{x}_0 \Leftarrow f\left(\sum_{i=0}^{8} \mathbf{W}_k \mathbf{x}_k\right)$, where $f(\cdot)$ is the element-wise non-linear function (Goodfellow et al., 2016). The adjacency matrix of a graph exhibits the local connectivities of the nodes. However, in contrary to image, we can not discriminate between the neighbouring nodes. Accordingly, in the GNNs the feature vector of a given node is updated as

$$\mathbf{x}_0 \Leftarrow f\left(\mathbf{W} \sum_{i=0}^{k} \mathbf{x}_k\right) , \tag{1}$$

where k is the number of neighbouring nodes and $\mathbf{W}$ is the weight matrix assigned to all the nodes. Suppose the graph is not labeled (neither the node nor the edges are labeled). Thus, with the feature aggregation (1) we only learn about the degree of the given node and in the subsequent message passing the information about the degrees of the nodes are propagated around the graph. There is no mechanism in this approach to learn the topological structure of the graph.
For instance, suppose we have a data-set of clustered unlabeled graphs (by unlabeled here we mean the nodes and the edges are not labeled) and assume that the clusters are not topologically different (a same generator created all the clusters). One class of the graphs consist of two clusters and the other class of graphs consist of three clusters. Consider the simplest case in which there is no connection between the clusters. In addition, assume that we use a typical GNN in which the feature propagation

(1) is performed a couple of times and subsequently all the feature vectors are aggregated using an indiscriminate aggregate function such as the element-wise max function or the element-wise mean function. Suppose that a given graph belongs to the first class, i.e., it consists of two clusters. Define $\mathbf{v}_1$ and $\mathbf{v}_2$ as the aggregated feature vectors corresponding to the first and the second clusters, respectively (the global feature vector of this graph is equal to the element-wise mean/max of $\mathbf{v}_1$ and $\mathbf{v}_2$). Clearly, $\mathbf{v}_2$ can be indistinguishable from $\mathbf{v}_1$ since the clusters are generated using the same generator. Therefore, the feature vector of the whole graph can also be indistinguishable from $\mathbf{v}_1$. The same argument is also true for a graph with three clusters. Accordingly, the representation obtained by the GNN is unable to distinguish the two classes. This example shows that the typical GNN can be unable to learn to solve even a simple graph classification problem. In Section 6.1, we provide two numerical examples to show this phenomena.

An important step in almost all the deep learning based language modeling methods is the embedding step which projects the words with similar meaning or with similar role to close data points in the continuous embedding space (Goldberg & Levy, 2014). Therefore, the next layers of the network learn to solve the given task by analyzing the distribution of the points given by the embedding layer and also their order in the sequence. In the GNNs, this important embedding layer is missing. If the nodes are not labeled, the network may not understand the difference between different nodes and it is also unaware of the location of the nodes in the graph.

In text data, each sentence is a sequence of words. Thus, if the sentence is represented using a graph, it is always a sequence of nodes where each node is connected to the next node. In addition, there is a fixed dictionary of words. Thus, the nodes of all the graphs which represent the sentences are sampled from a fixed dictionary of nodes. Neither of these two privileges are available in graphs. The structure of the graphs in a data-set are not necessarily the same and we generality cannot assume that all the graphs are built from a fixed dictionary of nodes. Therefore, in the GNN we basically can not have a fixed embedding layer as in the deep networks designed for text data. Therefore, in this paper we use a two step method in which the first step is graph embedding to obtain a representation of the graph in the continuous space. The second step learns to solve the given task by analyzing the geometrical distribution of the latent node representations given by the embedding step. The embedding step not only encodes the topological structure into the distribution of the points in the continuous space but also it assigns distinct representative point to each node. It makes the GNN aware of the differences/similarities between the nodes and the locations of the nodes in the graph.

The second main motivation of the proposed work is the lack of a pooling function in most of the existing GNNs (Gilmer et al., 2017; Hamilton et al., 2017; Dai et al., 2016; Duvenaud et al., 2015; Li et al., 2015). Using a pooling function, the GNNs would be able to learn hierarchical features. If the distribution of the feature vectors of the nodes represent the topological structure of the graph, the node sampling problem can be translated into a column/feature sampling problem (Halko et al., 2011; Deshpande & Rademacher, 2010).

4 Proposed Approach

Inspired by the success of deep networks in language modeling , we want to provide a geometrical representation of the graph to the deep network in which the different nodes are represented with different points, the nodes which are close on the graph or have similar structural role are represented with close data points, and each node is represented with a distinct point. In other word, we want to provide a geometrical representation of the graph to the deep network in which the topological structure of the graph has been encoded. According to the discussion in Section 3, we cannot include a fixed embedding stage in the structure of the network which can handle all the graphs in the given data-set. Thus, we use a graph embedding method to solve the embedding problem for each graph independently and obtain a representation of the graph in the continuous embedding space (in all the presented experiments, we use DeepWalk as the graph embedding algorithm). Accordingly, the input to the deep network is $\mathbf{X} \in \mathbb{R}^{n \times (d_1 + d_2)}$ which is the concatenation of the embedding vectors and the given nodes feature-vectors/labels (if there is any) and d_2 is the dimension of the embedding space used by the graph embedding algorithm. We build the proposed network using three main computation blocks:

- Initial Feature Transformation: This stage is composed of several fully connected layers. Each layer is composed of a linear transformation followed by Batch Normalization (Ioffe & Szegedy, 2015) and an element-wise non-linear function.
- Local Features Aggregation: Inspired by the local feature aggregation in the convolutional networks, this unit combines the feature vector of a node with its neighbouring nodes. The feature vector of the ith node in the lth layer of the network is updated as

$$\mathbf{x}_i^l \Leftarrow g\left(\{\mathbf{W}^l \mathbf{x}_k^l\}_{k \in \mathcal{I}_i}\right) , \tag{2}$$

  where $\mathbf{W}^l$ is the weight matrix assigned to all the nodes. The function g is the aggregate function. In this paper, we use the element-wise $\max$ function. In section 5, we will discuss about other choices of the aggregate function.
- Graph Pooling: One of the main challenges of extending the architecture of the CNNs to graphs is to define a pooling function for graphs. It is hard to measure how accurate a sub-sampled graph represents the topological structure of a given graph. It is not evident to measure how similar are two nodes based on their location and their connections in the graph. In contrary, it is simple to check how close are two vectors in the Euclidean space. Accordingly, since the distribution of the feature vectors represents the topological structure of the graph, we define the primary aim of the proposed pooling function to preserve the spatial distribution of the feature vectors of the nodes. The proposed pooling function down-samples the graph by a factor 2^z where z is an integer greater than or equal to 1. In this paper, we always set $z = 1$, i.e., each layer of graph pooling down-sizes the graph by a factor of 2. The proposed pooling function is detailed in Section 4.1. In contrary to the pooling layer in the CNNs, the way the proposed method down-sizes the graph is not predefined and it depends on the spatial distribution of the feature vectors in each layer of the network. Thus, the down-sizing is performed dynamically with respect to the last distribution of the feature vectors.
- Final Aggregator: After k steps of graph pooling, the given graph is down-sized to a graph of size $n/2^k$. In this paper, we use this function as the final aggregator to summarize all the feature vectors into a global representation of the graph.

Figure 1 shows the architecture of the network we used in all the presented numerical experiments.

Remark 1. *If in the given data edge attributes are available, the edge attributes can be used in the feature aggregation (2). For instance, if $\mathbf{y}_{i,k}$ is the attribute of the edge between the ith node and the kth node, we can concatenate $\mathbf{y}_{i,k}$ with $\mathbf{x}_k$ in (2) to use the edge features.*

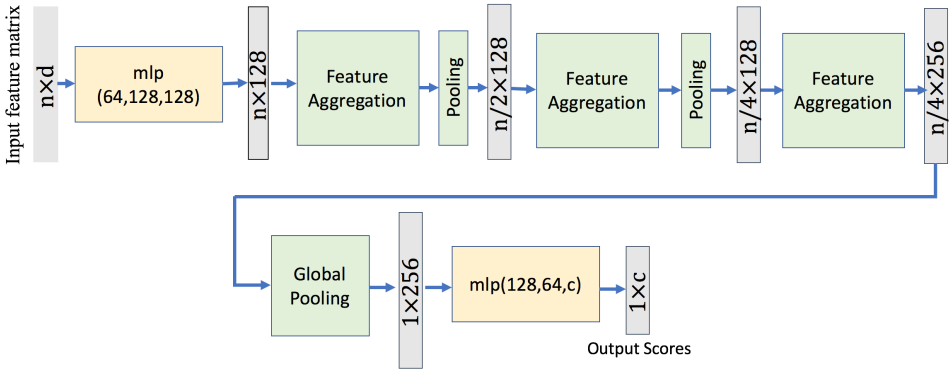

Figure 1: The architecture of the network we used in the numerical experiments. mlp stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp.

4.1 ADAPTIVE GRAPH POOLING

One of the main challenges of adopting the CNN structure in graph analysis is designing a pooling function applicable to graph data. Graphs lack the notion of spatial locality and the tensor repre-

sentation. It is not evident how to find a subset of the nodes such that they preserve the topological structure of the main graph. Accordingly, most of the existing GNNs are not able to perform the hierarchical feature learning as in the CNNs.

Although, it is hard to define a metric to measure how accurate a sub-set of the nodes preserve the topological structure of the graph, it is straightforward to define a metric to measure how accurate a subset of data points represent a set of given data points. The problem of sampling a sub-set of informative data points is a well-known problem in data analysis and it is mostly known as the column/row sampling problem (Halko et al., 2011; Deshpande & Rademacher, 2010; Drineas et al., 2006). However, most of the column sampling algorithms focus on finding a sub-set of the columns whose span yields an accurate low rank approximation of the given data.

In our application (graph down-sampling), instead of the row space of the data, we care about the spatial distribution of the feature vectors since they inherently exhibit the topological structure of the data. The Table of Algorithm 1 presents the proposed node/row sampling algorithm. In the proposed method, first the distance between the data points are computed. Subsequently, the closets pairs of points are found consecutively. Once the sets $\mathcal{A}$ and $\mathcal{B}$ are computed, the nodes in $\mathcal{A}$ are merged with the corresponding nodes in $\mathcal{B}$. The left image of Figure 2 shows how the ith node in $\mathcal{A}$ is merged with the corresponding node in $\mathcal{B}$. The set of neighbouring nodes of the result node is the union of the neighbour sets of the two nodes. In each graph pooling step, if we run Algorithm 1 z time, the size of graph is down-sampled by a factor of 2^z. In this paper, we set $z = 1$, i.e., each pooling layer down-sizes the graph by a factor of 2.

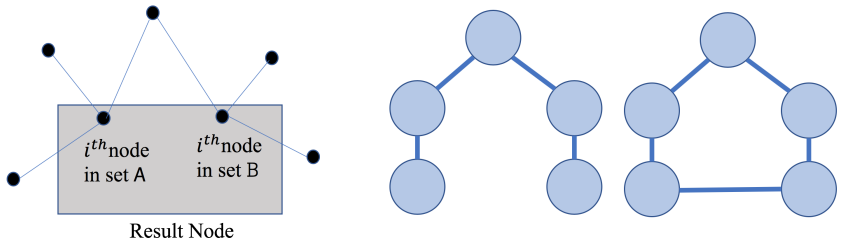

Figure 2: **Left**: Merging a pairs of nodes corresponding to the ith elements of $\mathcal{A}$ and $\mathcal{B}$. **Right:** This figure represents two graphs. The circles represent clusters (not nodes). Each cluster contains many nodes. The clusters in the right graph form a loop but in the left graph they do not.

Algorithm 1 Graph Down Sampling

Initialization: Define empty sets $\mathcal{A}$ and $\mathcal{B}$. Each node in $\mathcal{A}$ is merged with the corresponding node in $\mathcal{B}$.
Input: The matrix of feature vectors $\mathbf{X}^l \in \mathbb{R}^{n^l \times d^l}$ in the lth layer of the network.

1. Compute the ℓ_2-norms: Compute vector $\mathbf{q} \in \mathbb{R}^{n^l \times 1}$ such that $\mathbf{q}(k) = \|\mathbf{x}_i^l\|_2^2$. Define matrix $\mathbf{Q} \in \mathbb{R}^{n^l \times n^l}$ such that all the columns of $\mathbf{Q}$ are equal to $\mathbf{q}$.

2. Compute the distances: Compute $\mathbf{D} = \mathbf{Q} + \mathbf{Q}^T - 2\mathbf{X}^l{\mathbf{X}^l}^T$. $\mathbf{D}(i, j)$ is equal to the square of the distance between the ith and the jth row of $\mathbf{X}^l$.

3. Finding the closest pair of nodes:
3.1 For k from 1 to $\lfloor \frac{n^l}{2} \rfloor$
3.1.1 Define (i, j) as the location of the element of $\mathbf{D}$ with the minimum value.
3.1.2 Append i to $\mathcal{A}$ and j to $\mathcal{B}$.
3.1.2 Set the ith column, j^{th} column, i^{th} row, and j^{th} row of $\mathbf{D}$ equal to a big number not to sample these columns/rows in the next iterations.
3.1 End For
3.2 If n^l is an odd number, append the index of the remaining node to both set $\mathcal{A}$ and set $\mathcal{B}$.

4. Merging the pairs of nodes: The feature vector of the ith node in the down-sampled graph is equal to $\max(\mathbf{x}^l_{\mathcal{A}[i]}, \mathbf{x}^l_{\mathcal{B}[i]})$. The set of neighbouring nodes of the ith node in the down-sampled graph is the union of the set of neighbouring nodes of the corresponding pair of nodes.

5 FUTURE WORKS

In the proposed approach, each node of the graph is assigned a distinct data point in the continuous space and close nodes are represented with close data points. Although the graph lacks the tensor representation, we are aware of the positions of the nodes in the continuous space. Thus, inspired by the weighted feature aggregation in the CNNs, if $\mathbf{x}_i$ represent the feature vector of a given node and $\mathbf{x}_k$ is the feature vector of a neighbouring node, we can condition the weight matrix assigned to $\mathbf{x}_k$ on the $\mathbf{x}_i - \mathbf{x}_k$. Specifically, define function $\mathbf{H}(\cdot)$ such that its input is a vector and it outputs a weight matrix. Thus, we can aggregate the local features as $\mathbf{x}_i \Leftarrow \sum_{k \in I_i} \mathbf{H}(\mathbf{x}_i - \mathbf{x}_k)\mathbf{x}_k$. In order to build the weight matrix generator $\mathbf{H}(\cdot)$, we can implement a side network (which is trained with the main classifier) to generate the weights for the main network. In Ying et al. (2018), a similar idea was proposed to condition the weight matrices on the edge attributes. However, an edge attribute may not contain the information which can specify the relation of two neighbouring nodes. In addition, in most of the applications the edge attributes are not available or the graph is sparsely labeled.

6 NUMERICAL EXPERIMENTS

In this section, first we provide two experiments with synthetic data to show that the conventional GNNs can fail to infer the topological structure of the graphs. Subsequently, we apply the proposed approach to several established graph classification benchmark data-sets and compare the classification accuracy of the proposed approach against several kernel based methods and multiple recently proposed deep learning based approaches. In all the experiments, we used DeepWalk (Perozzi et al., 2014) to obtain the geometrical representation of the graphs. The architecture depicted in Figure 1 was used in all the experiments. We train the network using the implementation of Adam (Kingma & Ba, 2014) method in Pytorch. In all the experiments, we kept the learning rate fixed equal to 10^{-4}. In the proposed method, a dropout layer with dropout rate 0.6 is used after the last two dense layers. In addition to the performance of the proposed approach with graph pooling, we report the performance of the proposed approach without graph pooling too (the same architecture depicted in Figure 1 but with the pooling layers removed).

6.1 SIMPLE GRAPH ANALYSIS TASKS WITH SYNTHETIC DATA

In this section, we define two graph analysis tasks with synthetic data to show that the mere extension of the CNNs to graph data might fail to infer the structure of the graph. Thus, we study the performance of the proposed method and the network proposed in Zhang et al. (2018). In addition, we study a simple GNN which consists of 4 steps of message passing followed by an aggregate function (element-wise max function) applied to all the nodes. The generated graphs are composed of 3 to 6 clusters (the number of clusters are chosen randomly per graph). Each cluster is composed of 30 to 60 nodes (the number of nodes are chosen randomly per cluster). Each node in a cluster is connected to 5 other nodes in the same cluster. In addition, if we connect two clusters, we make 3 nodes of one cluster densely connected to 3 nodes of the other cluster.

6.1.1 DETECTING HIGH LEVEL LOOP

In this task, we trained the classifiers to detect if the clusters in a graph form a loop. Obviously, there are many small loops inside each clusters. Here we trained the algorithms to detect if the clusters form a high level loop. The right image of Figure 2 demonstrates the classification problem. We trained the three networks with 3000 training graphs and tested with 500 graphs. The accuracy of the proposed approach, DGCNN, and the simple GNN are 0.99, 0.55, 0.53, respectively. It is clear that DGCNN and the simple GNN failed to understand the structure of the graph.

6.1.2 COUNTING THE NUMBER OF CLUSTERS

We repeated experiment 6.1.1 but we trained the networks to estimate the number of the clusters. Thus, there are 4 classes of graphs. The accuracy of the proposed approach, DGCNN, and the simple GNN are 1, 0.35, 0.33, respectively. The important observation is that the mere extension of the CNNs to graphs can fail to infer the structure of the graph since they are missing an important embedding step.

Table 1: Comparison with the kernel based methods.

Data-set	MUTAG	PTC	PROTEINS	NCI1	DD B	IMDB
WL	84.11	57.97	74.68	**84.46**	78.34	72.86
RW	79.17	55.91	59.57	> 3 days	> 3 days	64.54
GK	81.39	55.65	71.39	62.49	74.38	65.87
GEO-DEEP	**92.22**	**73.76**	**79.1**	75.47	**78.87**	**73.2**
GEO-DEEP (no pooling)	91.66	73.53	78.01	75.79	78.38	71.2

6.2 Graph classification on Real Data

In this section, we study the performance of proposed approach (GEO-DEEP) on several established data-sets (Kersting et al., 2016). Following the conventional settings, we performed 10-fold cross validation, 9 folds for training and 1 fold for testing. Since most of the algorithms can not use the edge attributes, we did not include them in the experiments. We compare the proposed approach (GEO-DEEP) with three graph kernels, the graphlet kernel (GK) (Shervashidze et al., 2009), the random walk kernel (RW) (Vishwanathan et al., 2010), and the Weisfeiler-Lehman sub-tree kernel (WL) (Shervashidze et al., 2011) on six data-sets. Table 1 shows that on most of the data-sets the proposed approach outperforms the kernel based methods which are mostly computationally expensive. In addition, Table 2 compares the proposed approach against 6 recently published deep learning based graph classification methods DGCNN (Zhang et al., 2018), DIFFPOOL (Ying et al., 2018), PSCN (Niepert et al., 2016), DCNN (Atwood & Towsley, 2016), ECC (Simonovsky & Komodakis, 2017), and 2D-CNN (Tixier et al., 2018) on 7 data-sets. The proposed approach outperforms the other methods on 5 data-sets. For instance for the PTC data-set, its accuracy is almost 11 percent higher than the previous methods. In these data-sets, the size of most of the graphs are small. Thus, the proposed approach without the pooling layers achieves close classification accuracy. We conjecture that the pooling layer can yield more clear improvement for large size graphs.

Table 2: : Comparison with the other deep learning based approaches.

Data-set	MUTAG	PTC	PROTEINS	NCI1	DD	IMDB B	IMDB M
DIFFPOOL	-	-	78.10	-	**81.15**	-	-
ECC	76.11	-	-	**76.82**	-	-	-
2D-CNN	-	-	77.12	-	-	70.40	-
DCNN	0.8013	0.5530	-	0.6261	-	-	-
DGCNN	85.83	58.59	75.54	74.44	79.37	70.03	47.83
PSCN	88.95	62.29	75.00	76.34	76.27	71.00	45.23
GEO-DEEP	**92.22**	**73.76**	**79.1**	75.47	78.87	**73.2**	48.20
GEO-DEEP (no pooling)	91.66	73.53	78.01	75.79	78.38	71.2	**49.27**

7 Conclusion

We pointed this important fact out that the mere extension of the CNNs to graphs may not be able to infer the structure of the graph. In the proposed approach, the network is given the continuous representation of the graph in the embedding space and the GNN processes the graph as a set of data points in the embedding space. In addition, we addressed one of the challenges of extending the architecture of the CNNs to graphs by proposing a graph pooling method. The proposed pooling method merges the closest pairs of nodes (close in the spatial domain). We showed that the proposed approach outperforms most of the existing graph classification methods.

REFERENCES

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1993–2001, 2016.

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Data Mining, Fifth IEEE International Conference on*, pp. 8–pp. IEEE, 2005.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Diane J Cook and Lawrence B Holder. *Mining graph data*. John Wiley & Sons, 2006.

Siamak Zamani Dadaneh and Xiaoning Qian. Bayesian module identification from multiple noisy networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2016(1):5, 2016.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pp. 2702–2711, 2016.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

Amit Deshpande and Luis Rademacher. Efficient volume sampling for row/column subset selection. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pp. 329–338. IEEE, 2010.

Petros Drineas, Michael W Mahoney, and S Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-based methods. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pp. 316–326. Springer, 2006.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 869–877, 2018.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864. ACM, 2016.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53 (2):217–288, 2011.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

David Haussler. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. *arXiv preprint arXiv:1805.11921*, 2018.

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. URL `http://graphkernels.cs.tu-dortmund.de`.

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

Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483*, 2012.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023, 2016.

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

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pp. 5099–5108, 2017.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pp. 488–495, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

Martin Simonovsky and Nikos Komodakis. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017.

Antoine J-P Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Graph classification with 2d convolutional neural networks. 2018.

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, Jure Leskovec, Joseph M Antognini, Jascha Sohl-Dickstein, Nima Roohi, Ramneet Kaur, et al. Hierarchical graph representation learning with differentiable pooling. *CoRR*, 2018.

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