# CAPSULES GRAPH NEURAL NETWORK

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

The high-quality node embeddings learned by GNN have been applied to a wide range of node-based graph structured applications and some of them have achieved state-of-the-art (SOTA) performance. However, when applying node embeddings learned from GNN to generate graph embeddings, the scalar node representation typically used in GNN may not suffice to preserve the node/graph relationships, resulting in sub-optimal graph embeddings.

Inspired by the Capsule Neural Network (CapsNet) (Sabour et al., 2017), we propose the Capsules Graph Neural Network (CapsGNN), which adopts the concept of capsules to address the weakness in existing GNN-based graph embeddings algorithms. By extracting node features in the form of capsules, routing mechanism can be utilized to capture important statistic information at the graph level. As a result, our model generates multiple embeddings for each graph to capture significant graph properties from different aspects. The attention module incorporated in CapsGNN is used to tackle graphs with various sizes which also enables the model to focus on critical parts of the graphs.

Our extensive evaluations with 9 graph-structured datasets demonstrate that CapsGNN has a high potential for large graph data analysis and powerful capability in capturing macroscopic properties of the whole graph. It outperforms other SOTA techniques on several graph classification tasks.

## 1 INTRODUCTION

Graph Neural Network is a general type of deep-learning architecture that can be directly applied to structured data. These architectures are mainly generalized from other well-established deep-learning models like CNN (Krizhevsky et al., 2012) and RNN (Mikolov et al., 2010). In this paper, we mainly focus on Convolution-based Graph Neural Networks which attract increasing interest recently in graph analysis domain. Convolution operation can be embedded into Graph Neural Networks from both spectral and spatial perspective. Bruna et al. (2013) defines the convolution of the graph in the Fourier domain which needs to calculate the eigendecomposition of the graph Laplacian. This method is computationally expensive. Later, Henaff et al. (2015) introduces Chebyshev expansion of the graph Laplacian to avoid computing eigenvectors and Kipf & Welling (2016) proposes to do convolution within 1-step neighbor nodes to reduce the complexity. From the spatial perspective, Hamilton et al. (2017) and Zhang et al. (2018) propose to define a node receptive-field at first and do convolution within this field during which the information of each node as well as their neighbor nodes is gathered and new representation of each node is generated through an activation function. Both of these two perspective perform well in node representation learning and some variants (Velickovic et al., 2017) that developed based on this convolution idea have proven to achieve SOTA in various tasks.

The success of GNN has inspired many deep-learning-based approaches to leverage on nodes embeddings extracted by GNN to generate graph embeddings. For example, Simonovsky & Komodakis (2017) and Atwood & Towsley (2016) generate graph embeddings by summing nodes features. This operation is based on the assumption that the structure properties space is shared by nodes and graphs embeddings which might be impractical since graph embeddings should be able to capture more complicated and abstract area-based information. Zhang et al. (2018) models graphs as grid so that CNN can be applied. They fix the size of input graph data and fills this size with sorted nodes, these nodes are sorted based on learned nodes embeddings and then 1-D CNN is applied to perform classification. A common limitation in these approaches is that, they generate graph em-

beddings from nodes embeddings by simply detecting the presence of different structure features, which fails to preserve the specific information of these features such as connection, position and direction information.

This limitation is introduced by GNN which extract features in the form of scalar. It is especially problematic when GNN is applied to graph-based applications, where nodes features are used to generate subgraph or graph embeddings. To build high-quality graph embeddings, it is important to not only detect the presence of different structures around each node but also preserve their specific information such as position, direction, connection, etc. However, encoding these information in the form of scalar means activating points in a vector one-by-one which is exponentially less efficient than encoding them with distributed embeddings and this has been discussed in Sabour et al. (2017). To address the limitation mentioned above and inspired by CapsNet, we propose to extend scalar to vector during the procedure of applying GNN to graph embeddings. Compared with scalar-based neural network, vector-based neural network preserves more information of node/graph relationships. Besides, it models each graph with multiple embeddings and each embedding reflects graph properties from different aspects. This is more representative than only one embedding used in other algorithms. The technique for extracting features in the form of vectors is proposed in Hinton et al. (2011) and used in Sabour et al. (2017). This technique is mainly devised for image processing domain. In their work, the extracted vector feature is referred to as capsule (a group of neurons in neural network), so we follow the same notation in the following. Introducing capsules allows us to use routing mechanism to generate high-level features which we believe is a more efficient way for features encoding. Compared with max-pooling in traditional CNN (Krizhevsky et al., 2012), routing preserves all the information from low-level capsules and routes them to the closest high-level capsules. This is helpful to extract important statistic information. In graph processing domain, Verma & Zhang (2018) also introduce capsules. However, they extend scalar features to capsules in a feature-based manner. Besides, they apply dimension reduction between layers to compress capsules back to scalar features which defeats the purpose of having capsules in the first place.

In addressing the aforementioned limitations, we propose Capsule Graph Neural Network (Caps-GNN), a novel deep learning architecture, which is inspired by CapsNet, uses node features extracted from GNN to generate graph embeddings. In this architecture, each graph is modeled as multiple embeddings and each of these embeddings reflects different aspects of the graph properties. More specifically, basic features are extracted in the form of capsules through GNN and routing mechanism is applied to naturally generate high-level graph capsules as well as class capsules. During the procedure of generating graph capsules, an Attention Module is introduced to tackle graphs in different sizes and it assigns different importance to different capsules so that this model focuses on important parts of the graph. We validate the performance of generated graph embeddings on classification tasks over 9 datasets and we achieve SOTA results on 6 out of this 9 benchmark datasets. T-SNE (Maaten & Hinton, 2008) is used to visualize the learned graph embeddings and the results demonstrate that different graph capsules capture graph information from different aspects.

## 2 BACKGROUND

Here, we provide a brief introduction to the Graph Neural Network, routing mechanism in Capsule Neural Network (CapsNet) and Attention mechanism which are used in our architecture.

### 2.1 GRAPH

By definition, a weighted directed graph can be represented by $\mathcal{G} = (\mathbb{V}, \boldsymbol{X}, \boldsymbol{A})$ where $\mathbb{V} = \{v_1, v_2, ...v_n\}$ is the set of nodes and $\boldsymbol{A} \in \{0,1\}^{N \times N}$ is the adjacency matrix. If there is an edge from $v_i$ to $v_j$, then $A_{ij} = 1$ otherwise $A_{ij} = 0$. $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ represents the features of each node and $d$ is the number of feature channels.

### 2.2 GRAPH NEURAL NETWORK

In our work, we mainly focus on Graph Convolutional Networks (GCNs) (Kipf & Welling, 2016) which is one of the most widely used GNN architecture. At each layer of the GCN, the convolution operation is applied to each node as well as its neighbors and the new representation of each node is computed by an activation function. This procedure can be written as:

$$\boldsymbol{Z}^{l+1} = f(\boldsymbol{T}\boldsymbol{Z}^l\boldsymbol{W}^l) \tag{1}$$

where $\boldsymbol{Z}^l \in \mathbb{R}^{N\times d}$ represents nodes features at the layer $l$, $d$ represents the number of feature channels and $\boldsymbol{Z}^0 = \boldsymbol{X}$, $\boldsymbol{W}^l \in \mathbb{R}^{d\times d'}$ is a trainable weight matrix which serves as a channel filter, $f$ is a nonlinear activation function, $\boldsymbol{T} \in \mathbb{R}^{N\times N}$ is the information transform matrix and it is usually built based on adjacency matrix $\boldsymbol{A}$ for guiding the information flowing between nodes in graphs.

A complete GNN usually stacks $L$ layers to generate final nodes embeddings $\boldsymbol{Z}^L$. In the architecture proposed by Kipf & Welling (2016), at the $l$th layer of GNN, the extracted features of each node actually take all its adjacent nodes within $l$ steps into consideration. So $l$ can be considered as the size of the node receptive-field at this layer. This special property inspired us to use nodes features extracted from different layers to generate the graph capsules.

### 2.3 Capsule Neural Network

The concept of capsules is invented by Hinton's team (Hinton et al., 2011) and used recently in Sabour et al. (2017) and Hinton et al. (2018). CapsNet is designed for image features extraction which is developed based on CNN. However, unlike traditional CNN in which the presence of feature is represented with scalar value in feature maps, the features in CapsNet are represented with capsules (vectors). In Sabour et al. (2017), the direction of capsules reflects the properties of the features and the length of capsules reflects the probability of the presence of the features. The transmission of information between layers follows the dynamic routing mechanism. The specific procedure of dynamic routing can be found in Appendix A.

According to dynamic routing mechanism, parent capsules are generated based on votes from children capsules. Each children capsule is equally routed to all parent capsules at the first iteration, this procedure generates the prediction parent capsules, the relationship between each children capsule to each parent capsule is computed by the scalar product of these two capsules. In the next iteration, dynamic routing adjusts the contribution from children capsules to each parent capsule based on the scalar product result. After several iterations, each children capsule is assigned to a closer parent capsule. In this way, parent capsules are able to contain high-level statistic information inferred from children capsules. In our architecture, we use this mechanism to generate high-level graph capsules on the basis of node capsules extracted from GNN.

### 2.4 Attention Mechanism

Attention mechanism is widely applied in image processing (Zheng et al., 2017) and natural language processing domain (Gehring et al., 2016) where it is used to find the most relevant parts of input data to the task target. The main procedure of applying attention is: 1) defining an attention measure which is used to measure the relevance of each part of input data to the task target. 2) normalizing generated attention value. 3) scaling each part with normalized attention value.

In CapsGNN, we apply Attention mechanism for two purposes: 1) scaling each node of input graphs so that generated graph capsules from different graphs are comparable even though these graphs are vastly different in sizes. 2) guiding model to focus on more relevant important parts of graphs.

## 3 Capsules in Graph Neural Network

In this section, we outline our proposed architecture and show how it is used to generate high-level representative capsules for graphs which are then applied to graph classification tasks.

### 3.1 Architecture

Figure 1 shows a simple CapsGNN architecture. It consists of three main phases: 1) Basic node capsules extraction: GNN is applied to extract local vertices features with different receptive-field and then primary node capsules are built in this phase. 2) High level graph capsules extraction: Attention Module and routing are combined to generate multiple representative capsules for graphs.

3) Graph classification: Routing mechanism is applied again to generate final classes capsules for graph classification. We explain each phase in detail in the following.

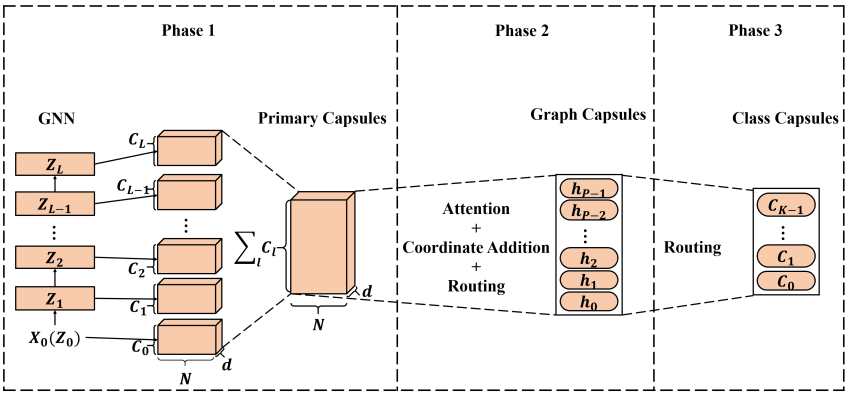



Figure 1: Framework of CapsGNN. At first, GNN is used to extract node embeddings and form primary capsules. Then, attention module is used to scale node embeddings which is followed by routing mechanism to generate graph capsules. At the last stage, routing is applied again to perform graph classification.

### 3.1.1 Basic Node Capsules

Firstly, the basic node features are extracted with GNN. Node degrees are used as node labels if nodes do not have labels. We use the architecture improved by Kipf & Welling (2016) (GCN) as the node features extractor. The difference is that we extract multi-scale nodes features from different layers and the extracted features are represented in the form of capsules. The procedure can be written as:

$$\boldsymbol{Z}_j^{l+1} = f(\sum_i \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \boldsymbol{Z}_i^l \boldsymbol{W}_{ij}^l) \tag{2}$$

where $\boldsymbol{W}_{ij}^l \in \mathbb{R}^{d \times d'}$ is the trainable weights matrix. It serves as the channel filters from the $i$th channel at the $l$th layer to the $j$th channel at the $(l+1)$th layer. Here, we choose $f(\bullet) = tanh(\bullet)$ as the activation function. $\boldsymbol{Z}^{l+1} \in \mathbb{R}^{N \times d'}$, $\boldsymbol{Z}^0 = \boldsymbol{X}$, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ and $\tilde{\boldsymbol{D}} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. To preserve structure features for sub-components with different sizes, we use nodes features extracted from all GNN layers as well as input layer to generate high-level capsules.

### 3.1.2 High-Level Graph Capsules

After getting local node-based capsules, global routing mechanism is applied to generate graph capsules. The input of this phase contains $N$ sets of node capsules, each set is $\mathbb{S}^n = \{\boldsymbol{s}_{11}, .., \boldsymbol{s}_{1C_1}, ..., \boldsymbol{s}_{LC_L}\}$, $\boldsymbol{s}_{lc} \in \mathbb{R}^d$, where $C_l$ is the number of channels at the $l$th layer of GNN, $d$ is the dimension of each capsule. The output of this phase is a set of graph capsules $\boldsymbol{H} \in \mathbb{R}^{P \times d'}$. Each of the capsules represents a type of graph property that is related to the graph classification task. The length of these capsules reflects the probability of the presence of these properties and the angle reflects the details of graph properties. Before generating graph capsules with node capsules, an Attention Module is introduced for scaling node capsules.

**Attention Module**. Primary capsules are extracted based on each node which means the number of primary capsules depends on the size of input graphs. However, the routing procedure introduced in Section 2.3 actually executes special summation over all children capsules. If the routing mechanism is directly applied, the value of generated high-level capsules will highly depend on the number of primary capsules (graph size) which is not the ideal case. Hence, an Attention Module is introduced to combat this issue.

The attention measure we choose is a two-layer fully connected neural network $F_{attn}(\bullet)$. The number of input units of $F_{attn}(\bullet)$ is $d \times C_{all}$ and the number of output units equals to $C_{all}$. We apply node-based normalization to generated attention value in each channel and then scale the

original nodes capsules. The details of Attention Module is shown in Figure 2 and the procedure can be written as:

$$scaled(\boldsymbol{s}_{(n,i)}) = \frac{F_{attn}(\tilde{\boldsymbol{s}_n})_i}{\sum_n F_{attn}(\tilde{\boldsymbol{s}_n})_i} \boldsymbol{s}_{(n,i)} \tag{3}$$

where $\tilde{\boldsymbol{s}_n} \in \mathbb{R}^{1 \times C_{all} d}$ is obtained by concatenating all capsules of the node $n$. $\boldsymbol{s}_{(n,i)} \in \mathbb{R}^{1 \times d}$ represents the $i$th capsule of the node $n$ and $F_{attn}(\tilde{\boldsymbol{s}_n}) \in \mathbb{R}^{1 \times C_{all}}$ is the generated attention value. In this way, the generated graph capsules is independent to the size of graphs and the model will focus more on important parts of input graphs.

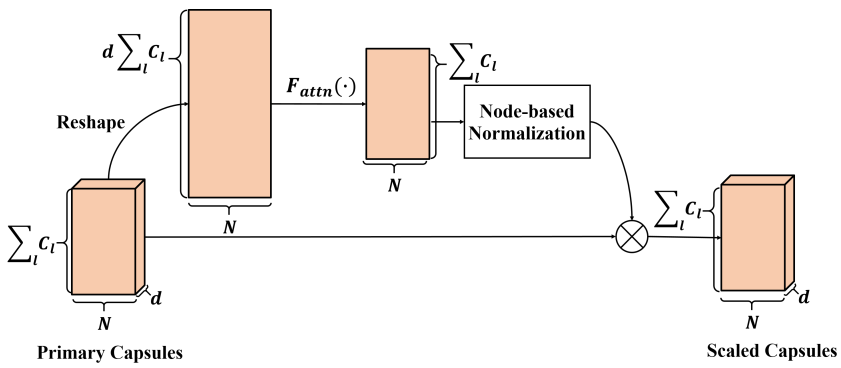



Figure 2: The structure of Attention Module. We first flatten primary capsules and apply two layer fully-connected neural network to generated attetion value for each capsule. Then, node-based normalization (normalize each column here) is applied to generate final attention value. Scaled capsules are calculated by multiplying the normalized value with primary capsules.

After Attention Module, during the procedure of generating node capsules votes, we use coordinate addition to preserve the position information of each node. When the GNN goes deeper, the extracted nodes features contain more specific position information. Therefore, we take the capsules extracted from the final layer of GNN as the position indicators of corresponding nodes by concatenating it with each capsule of the node.

The procedure of generating multiple capsules to capture various aspects of graph properties is as follows:

1) Scaling primary capsules: Applying Attention Module to scale primary capsules and the results of this module should be $\boldsymbol{S} \in \mathbb{R}^{N \times C_{all} \times d}$.

2) Calculating votes with coordinate addition: When calculating votes, capsules of different nodes from the same channel share the transform matrix. The result of this step is a set of votes $\boldsymbol{V} \in \mathbb{R}^{N \times (C_{all}-1) \times P \times d}$, the number of channels is $C_{all} - 1$ since we take one channel as position indicators. $P$ is the defined number of graph capsules.

3) Dynamic Routing Mechanism: High-level graph capsules are computed with the procedure introduced in Section 2.3 based on votes produced in previous steps.

#### 3.1.3 CLASSIFICATION

**Classification Loss**. Dynamic Routing is applied again over graph capsules to generate final class capsules $\boldsymbol{C} \in \mathbb{R}^{K \times d}$, where $K$ is the number of graph classes. Here, we use margin loss function proposed in Sabour et al. (2017) to calculate classification loss and it is computed as:

$$Loss_c = \sum_k \{T_k \max(0, m^+ - \|\boldsymbol{c}_k\|)^2 + \lambda(1 - T_k) \max(0, \|\boldsymbol{c}_k\| - m^-)^2\} \tag{4}$$

where $m^+ = 0.9$, $m^- = 0.1$ and $T_k = 1$ iff the input graph belongs to class $k$. $\lambda$ is used to stop initial learning from reducing the length of all class capsules especially when $K$ is large.

**Reconstruction Loss**. We use reconstruction loss as regularization method. Here, we mask all classes capsules except the most activate one and decode it with two fully-connected layer to reconstruct the input information. The information we reconstruct here is the histogram of input nodes. The procedure can be written as:

$$Loss_r = \frac{\sum_i MP_i(\boldsymbol{d}_i - \boldsymbol{m}_i)^2}{\sum_i MP_i} + \frac{\sum_i (1 - MP_i)(\boldsymbol{d}_i - \boldsymbol{m}_i)^2}{\sum_i (1 - MP_i)} \tag{5}$$

where $\boldsymbol{m}_i$ represents the number of nodes with the label $i$ appear in the input graph, $\boldsymbol{d}_i$ is the correspond decoded value. $MP_i = 1$ iff input graph contains nodes with the label $i$. This equation is used to prevent reducing reconstruction loss from setting all decoded value as 0.

### 3.2 Differences from other algorithms

The architecture details presented in section 3.1 describe key design idea of our proposed approach for graphs which is based on GNN and CapsNet. Here, we present a general comparison between CapsGNN with other related work.

1) Compared with Atwood & Towsley (2016), Simonovsky & Komodakis (2017) and Zhang et al. (2018) (GNN-based graph representation learning architectures), CapsGNN represents node features in the form of capsules. This is helpful to efficiently preserve the specific properties information contained in nodes especially when generating graph embeddings. Besides, each graph is modeled as multiple embeddings in CapsGNN instead of only one embedding used in other algorithms. This allows us to capture information of graphs from different aspects. The second difference is that, in these algorithms, each part of the graph is given equal importance. However, the attention mechanism used in CapsGNN allows it to assign different importance to different node. This leads the model to focus more on critical parts of input graphs. Lastly, different from Atwood & Towsley (2016) and Simonovsky & Komodakis (2017), CapsGNN and Zhang et al. (2018) uses node features extracted from multiple layers of GNN so that different size of receptive-fields are applied to preserve more information.
2) GCAPS-CNN proposed by Verma & Zhang (2018) also introduced capsules into graph representation learning. However, they generate capsules in a feature-based manner instead of learning graphs into distributed embeddings. More specifically, when they extend a scalar feature to a capsule for node $n$, they calculate $P$ higher-order statistical moment value based on its neighbor nodes and this $P$ value can be concatenated to a $P$-dimensional capsule. Between layers, they perform dimension reduction to compress capsules back to scalar features. CapsGNN learns each dimension of capsules in a data-driven manner and apply routing mechanism instead of dimension reduction between layers to preserve the learned meaning of each capsule. This also allows us to efficiently preserve multiple specific properties information contained in nodes especially when generating graph embeddings.
3) Compared with CapsNet proposed by Sabour et al. (2017) which works well in image processing domain, CapsGNN needs to handle more complicated situation. In the domain of image processing, the size of the input images can be standardized by resizing the images. However, it is not possible to simply resize the graphs. So, we introduced an additional Attention Module to tackle graphs that are vastly different in sizes and preserve more important parts of graphs. We also propose to make use of features extracted from all layers of GNN since it is hard to define a suitable receptive-field size for graphs. Furthermore, compared with the architecture of CapsNet, CapsGNN has one additional graph capsules layer which is used to learn multiple graph embeddings and these embeddings reflect different aspects of graph properties which is valuable in future research. To the best of our knowledge, we are the first one to model a graph as multiple embeddings in the form of distributed capsules and we believe this approach of learning representations has a high potential for other complex data analysis which is not limited for graphs.

## 4 Experiments

We verify the performance of graph embeddings extracted by CapsGNN against a number of SOTA and previous classical algorithms on classification task with 9 benchmark datasets. We also conduct

brief analysis on the generated graph capsules as well as the learned attention distribution. The experimental setting, results and analysis is shown in the following.

### 4.1 GRAPH CLASSIFICATION

The goal of graph classification is to predict the classes these graphs belong to by analyzing the structure and nodes labels information of graphs. More specifically, given a set of labeled graphs $\mathbb{D} = \{(\mathcal{G}_1, y_1), (\mathcal{G}_2, y_2), \dots\}$ where $y_i \in \mathbb{Y}$ is the label of each graph $\mathcal{G}_i$. The objective of graph classification is to find a mapping $f$ such that $f : \mathcal{G} \rightarrow \mathbb{Y}$.

### 4.2 DATASET

We choose 4 biological informatics graph datasets: MUTAG, NCI1, PROTEINS, D&D and 5 social network datasets: COLLAB, IMDB-B, IMDB-M, RE-M5K, RE-M12K (Yanardag & Vishwanathan, 2015). Details of these datasets can be found in Appendix B.

### 4.3 BASELINES METHODS

We compare CapsGNN with both kernel-based and other deep-learning-based algorithms:

**Kernel-based Methods**: We compare with 3 kernel-based algorithms: the Weisfeiler-Lehman subtree kernel (WL) (Shervashidze et al., 2011), the graphlet count kernel(GK) (Shervashidze et al., 2009), and Random Walk (RW) (Vishwanathan et al., 2010). Typically, kernel-based algorithms first decompose graphs into sub-components based on the kernel definition, then build graph embeddings in a feature-based manner. Lastly, some machine learning algorithms (i.e., SVM) are applied to perform graph classification.

**Deep-Learning-based Methods**: There are 3 types of deep-learning-based algorithms:

1) Graph2vec (Narayanan et al., 2017) and Deep Graph Kernel (DGK)(Yanardag & Vishwanathan, 2015). Both Graph2vec and DGK require extracting sub-structures in advance while Graph2vec learns the representation of graphs in the manner of Doc2vec (Le & Mikolov, 2014), DGK applies Word2vec (Mikolov et al., 2013) to learn the similarity between each pair of sub-structures which will be used to build graph kernel. Then kernel-based machine learning methods (i.e., SVM) are applied to perform graph classification. These algorithms as well as kernel-based methods are all sub-components based which require computationally intensive preprocessing effort. Although Graph2vec and DGK also apply learning approaches to learn the embeddings, they still need to extract sub-components before learning procedure. So we consider Graph2vec, DGK and other kernel-based algorithms as the same type in our experiments.

2) PATCHY-SAN (Niepert et al., 2016). This method first sorts all nodes, then defines a receptive-field size for each node. These receptive-field are then filled with sorted neighbor nodes. Lastly, 1-D CNN is applied to perform graph classification.

3) GCAPS-CNN (Verma & Zhang, 2018), Dynamic Edge CNN (ECC) (Simonovsky & Komodakis, 2017) and Deep Graph CNN (DGCNN) (Zhang et al., 2018). These methods are all GNN-based algorithms. GCAPS-CNN first extract FGSD (Verma & Zhang, 2017) features for nodes without labels and then generate capsules for each center node with higher-order statistical moment value of its neighbor nodes. At the last layer, they calculate covariance between all nodes to generate graph embeddings. ECC extracted node features on the condition of edge labels in GNN and then apply multi-scale pyramid structure to coarsen graph. They use average pooling at the last layer to generate graph embeddings. DGCNN is the most similar one to ours. They also generate nodes embeddings through a multi-layer GNN and combine features extracted from all layers. Then they order all the nodes based on the embeddings extracted from the last layer of GNN which is followed by 1-D CNN.

### 4.4 EXPERIMENTAL SET-UP

We use the same architecture setting for all datasets to show its robust performance. For the basic node capsules extraction, the GNN has 5 layers ($L = 5$), the number of channels at each layer is set as the same which is 2 ($C_l = 2$). The number of graph capsules is fixed as 16 ($P = 16$).

The dimension of all capsules are all set as 8 ($d = 8$). The number of units in the hidden layer of Attention Module is set as $\frac{1}{16}$ of the number of input units. The number of iterations in routing is set as 3. During training stage, we simultaneously reduce $Loss_c$ and $Loss_r$ and we scale $Loss_r$ with $0.01$ so that the model focuses on classification task. $\lambda$ is set as $0.5$ and $1.0$ for multi-class classification and binary classification respectively.

To evaluate the performance objectively, we applied 10-fold cross validation on each dataset. Each time we use 1 training fold as validation fold to adjust hyper-parameters, 8 training fold to train weighs and the remained 1 testing fold to test the performance. We stop training when the performance on validate fold start to reduce. Then we use the accuracy on test fold as our test result. The final result is the average of these 10 test accuracy. By default, we use the results reported in their original work for baseline comparison. However, in cases where the results are not available, we use the best testing results reported in Verma & Zhang (2018) and Zhang et al. (2018).

### 4.5 Classification Result

Table 1 lists the results of the experiments on biological datasets and Table 2 lists the results of the experiments on social datasets. For each dataset, we highlight the top 2 accuracy in bold. Compared with all the other algorithms, our proposed architecture can achieve top 2 on all the social dataset and achieve comparable results on biological dataset except D& D. Compared with all the other deep-learning based algorithms, CapsGNN achieve top 1 on all social datasets.

Table 1: Experiment Result of Biological Dataset

| **Algorithm** | **MUTAG** | **NCI1** | **PROTEINS** | **D&D** |
|---|---|---|---|---|
| WL | $84.11 \pm 1.91$ | $\mathbf{84.46 \pm 0.45}$ | $74.68 \pm 0.49$ | $\mathbf{78.34 \pm 0.62}$ |
| GK | 84.04 | $62.49 \pm 0.27$ | $71.39 \pm 0.31$ | $74.38 \pm 0.69$ |
| RW | $79.17 \pm 2.07$ | $> 3days$ | $74.22 \pm 0.42$ | $> 3days$ |
| Graph2vec | $83.15 \pm 9.25$ | $73.30 \pm 2.05$ | $73.22 \pm 1.81$ | $71.51 \pm 4.02$ |
| DGK | $\mathbf{87.44 \pm 2.72}$ | $80.31 \pm 0.46$ | $\mathbf{75.68 \pm 0.54}$ | $73.50 \pm 1.01$ |
| PSCN | $88.95 \pm 4.37$ | $76.34 \pm 1.68$ | $75.00 \pm 2.51$ | $76.27 \pm 2.64$ |
| DGCNN | $85.83 \pm 1.66$ | $74.44 \pm 0.47$ | $75.54 \pm 0.94$ | $\mathbf{79.37 \pm 0.94}$ |
| ECC | – | 76.82 | 72.65 | 74.1 |
| Caps-CNN | – | $\mathbf{82.7 \pm 2.38}$ | $\mathbf{76.40 \pm 4.17}$ | $77.62 \pm 4.90$ |
| **CapsGNN** | $\mathbf{90.44 \pm 4.58}$ | $79.42 \pm 2.30$ | $75.20 \pm 4.83$ | $74.89 \pm 4.38$ |

Table 2: Experiment Result of Social Dataset

| **Algorithm** | **COLLAB** | **IMDB-B** | **IMDB-M** | **RE-M5K** | **RE-M12K** |
|---|---|---|---|---|---|
| WL | $\mathbf{79.02 \pm 1.77}$ | $\mathbf{72.86 \pm 0.76}$ | $\mathbf{50.55 \pm 0.55}$ | $49.44 \pm 2.36$ | $38.18 \pm 1.3$ |
| GK | $72.84 \pm 0.28$ | $65.87 \pm 0.98$ | $43.89 \pm 0.38$ | $41.01 \pm 0.17$ | $31.82 \pm 0.08$ |
| DGK | $73.09 \pm 0.25$ | $66.96 \pm 0.56$ | $44.55 \pm 0.52$ | $41.27 \pm 0.18$ | $32.22 \pm 0.10$ |
| PSCN | $72.60 \pm 2.15$ | $71.00 \pm 2.29$ | $45.23 \pm 2.84$ | $49.10 \pm 0.70$ | $\mathbf{41.32 \pm 0.32}$ |
| DGCNN | $73.76 \pm 0.49$ | $70.03 \pm 0.86$ | $47.83 \pm 0.85$ | $48.70 \pm 4.54$ | – |
| Caps-CNN | $77.71 \pm 2.51$ | $71.69 \pm 3.40$ | $48.50 \pm 4.10$ | $\mathbf{50.10 \pm 1.72}$ | – |
| **CapsGNN** | $\mathbf{78.92 \pm 1.89}$ | $\mathbf{72.70 \pm 4.05}$ | $\mathbf{49.93 \pm 3.50}$ | $\mathbf{52.88 \pm 1.48}$ | $\mathbf{46.62 \pm 1.90}$ |

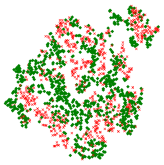
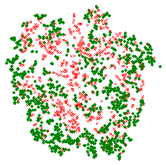
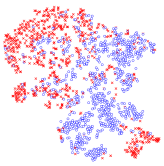
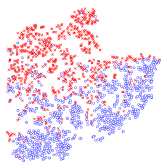
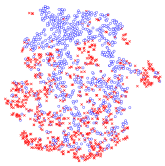
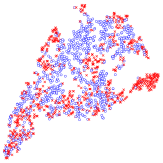
Our proposed architecture achieves the SOTA performance on Social datasets. More specifically, we are able to improve the classification accuracy by a margin of 1.7% and 5.3% on RE-M5K and RE-M12K respectively. This demonstrates that learning features in the form of capsules and modeling

a graph to multiple embeddings is beneficial to capture macroscopic properties of graphs. These macroscopic properties are more important in classifying social networks. This results also agree with the property of CapsNet that focuses more on extracting important statistic information from children capsules. However, this property is not suitable to preserve information of fine structures which might be more important to biological datasets analysis. This cause the worse performance of CapsGNN on biological datasets. Despite this, the performance of CapsGNN in graph classification tasks still demonstrates its capability of graph representation and its high potential of large graph data analysis.

### 4.6 Graph Capsules Analysis

CapsGNN models each graph as multiple embeddings to capture complex information underlying graphs. To explore the properties of the extracted graph capsules, we separately plot capsules extracted from different channels with t-SNE (Maaten & Hinton, 2008) and the results are shown in Table 3. Due to space constrain, we only take Reddit-12K as an example. Here, we show that different graph capsules capture information from different aspects. Each graph in REDDIT dataset represents a discussion thread in which each node indicates a user and there is a link between two users if at least one of them responded to another's comment. The task is to predict the subreddit where the given discussion graph belongs. Here, we choose to depict the distribution of graphs which are generated from *atheism* , *IAmA* and *mildlyinteresting* with capsules extracted from the 1st, 2nd, 11th and the 14th channel. Similar phenomenon can also be observed in other datasets.

Table 3: Visualization of Capsules

| **Subreddit** | **Channel 1** | **Channel 2** | **Channel 11** | **Channel 14** |
|---|---|---|---|---|
| atheism & IAmA |  |  | | |
| atheism & mildlyinteresting |  |  |  |  |

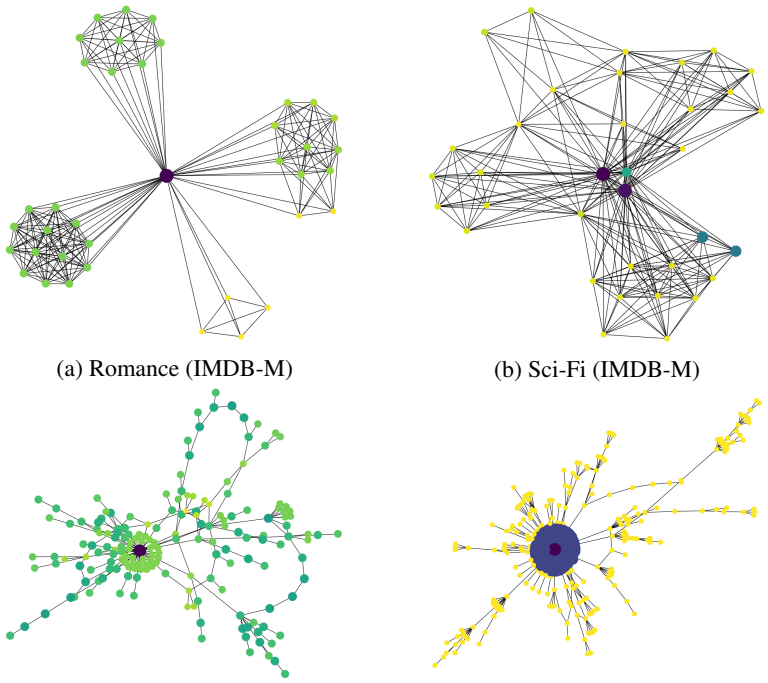
As we can see, different channels of capsules represent different aspects of graph properties. *atheism* and *IAmA* can be discriminated obviously with capsules extracted from the 11th and the 14th channels while they are hard to be separated with capsules extracted from the 1st and the 2nd channels. However, *atheism* and *mildlyinteresting* can be discriminated with the capsules extracted from the 1st and the 2nd channels while they are mixed in the 11th and the 14th channel which is opposite to the case of *atheism* and *IAmA*. This phenomenon can also be observed in other multi-class datasets. It is still hard to figure out the specific aspects these capsules focus on, however, compared with scalar-based neural networks, modeling an objective as multiple embeddings makes it possible to explore the meaning of each channel and lead the model to learn more interpretable embeddings in the future research.

### 4.7 Attention Analysis

We conduct node-based normalization in Attention Module to scale different node capsules to force the model focusing on critical parts of each graph. In this experiment, we visualize the attention distribution on REDDIT and IMDB-M dataset to see if this model focuses on critical parts of graphs. For IMDB-M dataset, graphs in IMDB-M are all ego-networks derived from movie-collaboration relationship networks in which each node represents an actor/actress and there is a link between two

nodes if they appeared in the same movie. These collaboration networks are built based on movies in different genres namely *Romance*, *Sici-Fi* and *Comedy* and the task for IMDB-M dataset is to predict the genre the given graph belongs to.

The visualization results are shown in Figure 3 where we can find that the model always pays more attention to nodes with high centrality or critical nodes that connect multiple separate sub-components of the graph which makes sense as critical parts of graphs from common sense.



(a) Romance (IMDB-M)

(b) Sci-Fi (IMDB-M)

(c) atheism (REDDIT)

(d) AskReddit (REDDIT)

Figure 3: Visualization of attention distribution on IMDB-M dataset and REDDIT dataset. Nodes with higher attention are darker in color and larger in size.

## 5 Conclusion

We have proposed CapsGNN, a novel architecture that addresses the weakness of existing GNN approaches in the literature. Inspired by CapsNet, the concepts of capsules are introduced in this architecture to extract features in the form of vectors on the basis of nodes features extracted by GNN. As a result, one graph is modeled as multiple embeddings and each of these embeddings captures different aspects of the graph properties. This is a novel and powerful data-driven method to represent high-dimensional data such as graphs. With the help of the introduced Attention Module, CapsGNN can tackle graphs that are vastly different in sizes and can focus on critical parts of graphs. Our model has successfully achieved better or comparable performance when compared with other SOTA algorithms on 6 out of 9 graph classification tasks especially on social datasets. This results show that learned capsules can capture more macroscopic information of graphs.

## References

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

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

Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*, 2016.

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

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

Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pp. 44–51. Springer, 2011.

Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. 2018.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pp. 1188–1196, 2014.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *CoRR*, abs/1707.05005, 2017. URL `http://arxiv.org/abs/1707.05005`.

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

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pp. 3856–3866, 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.

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

Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*, pp. 88–98, 2017.

Saurabh Verma and Zhi-Li Zhang. Graph capsule convolutional neural networks. *arXiv preprint arXiv:1805.08090*, 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.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374. ACM, 2015.

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.

Heliang Zheng, Jianlong Fu, Tao Mei, and Jiebo Luo. Learning multi-attention convolutional neural network for fine-grained image recognition. In *Int. Conf. on Computer Vision*, volume 6, 2017.

## A SPECIFIC PROCEDURE OF ROUTING

The specific procedure of routing is shown in Algorithm 1.

**Algorithm 1** Dynamic routing mechanism returns parent capsules $\boldsymbol{H}$ given children capsules $\boldsymbol{S}$, a set of trainable transform matrices $\mathbb{W}$ and the number of iterations $t$.

**procedure** DYNAMIC ROUTING($t, \boldsymbol{S}, \mathbb{W}$)
  for all children capsule $i$ : $\boldsymbol{v}_{j|i} = \boldsymbol{s}_i^T \boldsymbol{W}_{ij}$
  for all children capsule $i$ to all parent capsule $j$: $r_{ij} \leftarrow 0$
  **for** $t\ iterations$ **do**
    for all children capsule $i$: $\tilde{\boldsymbol{r}}_i \leftarrow softmax(\boldsymbol{r}_i)$
    for all parent capsule $j$: $\boldsymbol{h}_j \leftarrow \sum_i \tilde{r}_{ij} v_{ij}$
    for all parent capsule $j$: $\tilde{\boldsymbol{h}}_j \leftarrow squash(\boldsymbol{h}_j)$
    for all children capsule $i$ to all parent capsule $j$: $r_{ij} \leftarrow r_{ij} + \tilde{\boldsymbol{h}}_j^T \boldsymbol{v}_{ij}$
  **end for**
  **return** $\tilde{\boldsymbol{h}}_j$
**end procedure**

## B DETAILS OF EXPERIMENTAL DATASETS

The details of benchmark datasets we use in our experiment is shown in Figure 4.

Table 4: Dataset Description

| Dataset | Source | Graphs | Classes | Nodes Avg. | Edges Avg. | Nodes Labels |
|---|---|---|---|---|---|---|
| MUTAG | Bio | 188 | 2 | 17.93 | 19.79 | 7 |
| NCI1 | Bio | 4110 | 2 | 29.87 | 32.30 | 23 |
| PROTEINS | Bio | 1113 | 2 | 39.06 | 72.81 | 4 |
| D& D | Bio | 1178 | 2 | 284.31 | 715.65 | 82 |
| COLLAB | Social | 5000 | 3 | 74.49 | 4914.99 | - |
| IMDB-B | Social | 1000 | 2 | 19.77 | 193.06 | - |
| IMDB-M | Social | 1500 | 3 | 13 | 131.87 | - |
| REDDIT-M5K | Social | 4999 | 5 | 508.5 | 1189.74 | - |
| REDDIT-M12K | Social | 11929 | 11 | 391.4 | 456.89 | - |