CAPSGNN REIMPLEMENTATION REPORT

Yuzhe Yang, Yuqian Dai & Zheng Lyu
School of Electronics and Computer Science
University of Southampton
Southampton, UK
`{yy1a19,yd6u19,zl9y19}@soton.ac.uk`

ABSTRACT

We choose CapsGNN Xinyi & Chen (2018) as our reproducibility challenge. This network is proposed to cope with the low efficient and sub-optimal graph embedding problems in traditional graph neural network. This report contains three sections. First, we give a brief introduction about CapsGNN and its related components. Second we would reimplement the CapsGNN and test its performance and make comparisons with the given results. Then, we summarize a bunch of reflections related to the model performance and the characteristics of training CapsGNN. The results indicate that the proposed model can be basically reimplemented based on the provided code and the CapsGNN can reach a SOTA performance on social datasets. All the codes have been pushed on a github repository `https://github.com/YuzheYang/COMP6248-Reproducibility-Challenge-CapsGNN-TF`.

1 INTRODUCTION

The CapsGNN contains three components. CapsNet(Capsule Network), GCN(Graph Convolutional Network) and attention mechansim.

- **CapsNet:** Capsule Network use Capsules instead of CNN kernel for convolution. Capsules are different from traditional scalar neurons. Equation 1Sabour et al. (2017) is the dynamic routing coefficient calculation. The routing coefficient c_{ij} is determined by b_{ij} entering Softmax, and at the same time, $b_{ij}$is determined by the routing consistency a_{ij}. $v_j$is the output vector of the capsule of the previous layer and also the prediction vector of this layer. Routing consistency can be seen as a way to calculate the similarity of two vectors, thereby estimating the coupling coefficient. The capsule vectors in this layer will be connected to the higher-level vectors.

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \tag{1}$$

The architecture of the capsule network usually contains a standard convolutional layer with the ReLU activation function. Capsule's vector can represent an instance, and a specific vector will construct a complete image through two fully connected layers.

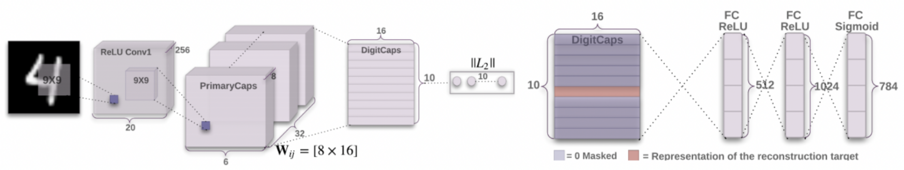

Figure 1: General capsule neural network for image classification Sabour et al. (2017)

- **GCN:** Graph Convolution Network could extract powerful feature without training Kipf & Welling (2016). Equation 2 is the convolution process of GCN. where $\hat{A} = A + I$, $\tilde{A}$

is the adjacency matrix, $H^{(l)}$ is the feature matrix of layer l. σ is a non-linear activation function. $\tilde{D}$ is the degree matrix of $\tilde{A}$.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2}$$

- **Attention Mechanism:** Attention mechanism is used to search specific area in an the input area. Here, it is used for scaling different input graph capsule and target on the specific area [1].

2 Experiment

In this work, we are going to reimplement the CapsGNN model based on the open-sourced code written with TensorFlow Xinyi & Chen (2018). As the author mentioned in their paper, the whole process contains two parts. The first thing is to re-organize and preprocess our structured data. The data sets gexf formation. Hence, we need to reorganize it into our format which can be utilized by GraphDataset. The secondary is to build up CapsGNN model and feed the preprocessed data into the model to achieve our goals.

2.1 Implementation

Literally, ten well-structured data sets are included in this experiment according to this paper. Therefore, the data generator need to be a universal framework so that we can scan and reconstruct different types of data easily. The data structure contains several attributes, such as number of edges, number of nodes and etc. .We could also link the graph to label so that this is a typical supervised learning process. Moreover, we reorganize the data into a dictionary class which contains three components, label, degree and constant. Since the size of our data sets are relatively small, we use ten-folder cross validation to measure the model performance and we apply this model on classification data sets and use classification accuracy and we use the test accuracy to measure our model performance. So we split the data into three groups, training set, validation set, and test set and make it into ten-fold cross-validation formation.

We first check the statistical information generated from the datasets. In this reimplementation work, we choose to conduct four of those 10 experiments to prove the related experiment results due to our limited computational resource. Here, two datasets are from biological datasets, which are NCI1 and ENZYMES. In addition, the rest two are from social datasets, which are IMDB-BINARY and IMDB-MULTI. We use pickle v3 to load data and we set the number of folds as 10. The basic dataset statistical information is listed below.

Table 1: Dataset Information

Dataset	**ENZYMES**	**CSI1**	**IMDB-BINARY**	**IMDB-MULTI**
Average number of edges	62.14	32.3	96.53	65.94
Average number of nodes	32.63	29.87	19.77	13.00
Graph number	600	4110	1000	1500
Number of Classes	6	2	2	3
Reconstruct index	0 2	0 2	0 2	0 2
Input index	0 1 2	0 1 2	0 1 2	0 1 2

From table 1, it can be observed that the related information are consistent. Hence, we can say that we have reimplemented and proved the preprocess section. After we preprocess the dataset and make sure that our data set is preprocessed correctly, we can then feed the data into the CapsGNN model. In the next section, make sure that the CapsGNN is the same as what they show in the paper.

2.1.1 Network structure checking

The code related to CapsGNN model gets 14 parameters as input, such as the node attributes, number of classes, learning rate and etc. The network structure contains three stages. The first stage is to use GNN to extract node embeddings and put the related embeddings into a primary capsule for further processing. First, we build up the node structure and adjacency matrix. Then, a function named node embedding generator is used to generated the embedded nodes with the information: node embeddings, transpose matrix, input channel, output channel and embedding size and etc. The output size here is (batch size, N, channel out and d out) and the embedding process is first we reshape the input with the shape (input shape [0], N, input channel and embedding node size). Then we pass the input into a 2D convolution with a node filter. Then we squeeze the output and multiply the output with transpose matrix and activate the result with tanh activation function and then reshape it again and return it back as the final result for stage 1. The related output is named as ”nets”.

In the second stage, if we set the attention config into True, then the output from stage 1 needs fed into an attention layer for scaling each node capsule so that the graph capsules are still comparable. The inside of the attention layer is that first we reshape the input into 1 dim and then we use two fully-connected layers to extract features and we use SoftMax to obtain the attention weights for scaling the input and the scaled inputs are multiplied with the number of nodes and return as the output of attention layer. Then the output from attention or directly from stage 1 without using attention module will be sent to a graph capsule which the dynamic routing generated from graph capsules. The capsule is first proposed by Hinton et al Sabour et al. (2017) and the capsule neural network here is the same as it is in Sabour et al. (2017). The output of capsule in the second stage is to use graph capsules to generate graph capsule features to make a further classification.

In the third stage, there is still a capsule graph neural network which is used to make classification. The classification capsule is to vote on the inputs and squeeze the result into a routing process and return two thing v_j and a_j for computing the loss and back propagation, where v_j is used for computing the reconstruction loss and a_j is used for computing the margin loss for calculating the classification error.

So far, we have got through the whole network structure and the result indicates that the whole process is the same to the explanations which are given by the author. In conclusion, we can say that the network structure has been proved and since we can understand the code and network structure clearly, this work should be able to be reimplemented. In the next section, we are going to check training process

2.1.2 TRAINING CHECKING

The training process contains three parts, which are training section, validate section and test section. The process on those three are generally similar, while the only change is that the model parameters will be frozen when it is going to the validation and test. The main result to prove the model performance is using classification accuracy. Hence, we reimplement the classification accuracy and add it into the network structure and add it into the training process, since the code related to this part is not given by the author. The training process contains many hyper-parameters. To be consistent with the experimental setting in the paper, we choose to use the released default parameters.

With that parameter setting, we retrain the model to measure the performance on four datasets. The related training results are shown in table 2.

Table 2: Parameters Setting

Dataset	**Loss**	**Error**	**Accuracy**	**Accuracy(Paper)**
NCI1	20.36	21.65	78.42	78.35 ± 1.55
ENZYMES	34.88	46.67	52.68	54.67 ± 5.67
IMDB	19.30	20.88	79.13	73.10 ± 4.83
IMDB-multi	28.96	54.00	46.82	50.52 ± 2.65

From the related results it can be observed that the model performance reimplemented by ourselves and the performance given by the author are very close. Therefore, we can say that we have successfully reimplement the training process based on the open-sourced code and the model performance provided by the author is reliable and it has proved that the CapsGNN can actually play well on those data and the performance on social dataset is better than its on biological datasets, which is in line with the conclusion given by the author. Hence, so far, we have reimplement the whole training and testing process and the conclusion is that the result given by the author is reliable and reproducible.

3 Reflection and Analysis

In this work, we have successfully reimplemented the works CapsGNN with the open-sourced code. To summarize, there are 4 points that we would like to reflect. First, the related source code has been proved that it is executable and the code structure is well-organized. In addition, the basic information of the data set and the training results are well-checked and they are in line with the given values in that paper. Third, the training process takes too much time and the learning efficiency of the capsule is still remaining to be improved. In fact, with the given parameters in that paper, it will take around 2 to 3 days to test one dataset with a single GPU NVIDIA RTX-2060 and Intel i7-9750H 2.6GHz. However, the model performance on the test set won't change a lot after it reaches around 900 epochs. Hence, in this reimplementation, we modify the epoch from 3000 to 1000 and the results indicate that there is not a great difference between these two. Fourth, during the training process, the loss fluctuation is fierce and declines slowly in the early stage of training. In addition, the model contains too much parameters and hyper-parameters which will take a long time to change and the model performance is largely influenced by the setting.

In addition, the paper says that the final test accuracy on those 10 datasets are based on the average value of 10-fold accuracy. However, when we are reimplementing, there is no such setting in the open-sourced code and there is no such a variable to calculate classification accuracy. Hence, we re-write the accuracy based on the source code and all of the related accuracy results given in this report is calculated by ourselves.

We failed to reimplement the t-SNE visualization of the graph capsule distribution and the classes of capsule. The original motivation for this section in the paper CapsGNN Xinyi & Chen (2018) is to help discovering the learnt features from different layers of capsules and prove that they all learnt different features for making the final classification. However, the explanations here in the paper are not quite clear and the open-sourced code does not include anything related to that part. Also, as we know the final visualization results are largely depended on the parameter setting in t-SNE. However, the author does not make any clarification related to the t-SNE parameter settings on those four datasets, which are tested in this reimplementation. Therefore, it is hard to reimplement this part. However, the author does implement some related visualizations and more details can be found at emb Therefore, the related reimplementation for this part could be conducted in the future.

4 Conclustion

In conclusion, we have passed the reimplementation successfully and we also have some related findings. Through reimplementing this experiment, our group member can understand the CapsGNN deeply and be clearer about the implementation details. Moreover, we could find out that the performance of graph neural network with capsule layers in it could improve its original performance and the vector features can store more spatial correlation information which is not included in the traditional convolution neural network. Technically, if we could discover the missing part related to the visualization work, we can get in touch with the author early to obtain the code related to those part so that we can analyze our work more in detail.

References

Embeddings visualization. URL `https://sites.google.com/view/capsgnn/embeddings-visualization`.

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

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pp. 3856–3866, 2017.

Zhang Xinyi and Lihui Chen. Capsule graph neural network. 2018.