GRAPH CLASSIFICATION USING MULTI-SCALE NODE FEATURES

Md Asadullah Turja **Jennifer Chen** **David Bang** **Kandan Sudhakar** **Filipp Gusev**

May 7, 2019

1 Introduction

The recent success of neural networks has boosted research on pattern recognition and data mining. Many machine learning tasks such as object detection [1], [2], machine translation [3], and speech recognition [4], which once heavily relied on handcrafted feature engineering to extract informative feature sets, has recently been revolutionized by various end-to-end deep learning paradigms, i.e., convolutional neural networks (CNNs) [5], long short-term memory (LSTM) [6], and autoencoders.

While deep learning has achieved great success on Euclidean data, there is an increasing number of applications where data are generated from the non-Euclidean domain and need to be effectively analyzed. For instance, in e-commerce, a graph-based learning system is able to exploit the interactions between users and products [7], [8] to make highly accurate recommendations. Graph data have been informative resources in exploring the structure of various objects, e.g. molecules and their biological activities, brain networks, interpersonal networks, and so on. In recent years there has been a surge of interest in developing graph neural networks (GNNs)— general deep learning architectures that can operate over graph structured data, such as social network data [9, 7] or graph-based representations of molecules [10]. The general approach with GNNs is to view the underlying graph as a computation graph and learn neural network primitives that generate individual node embeddings by passing, transforming, and aggregating node feature information across the graph [11]. The generated node embeddings can then be used as input to any differentiable prediction layer, e.g., for node classification [11] or link prediction [12], and the whole model can be trained in an end-to-end fashion.

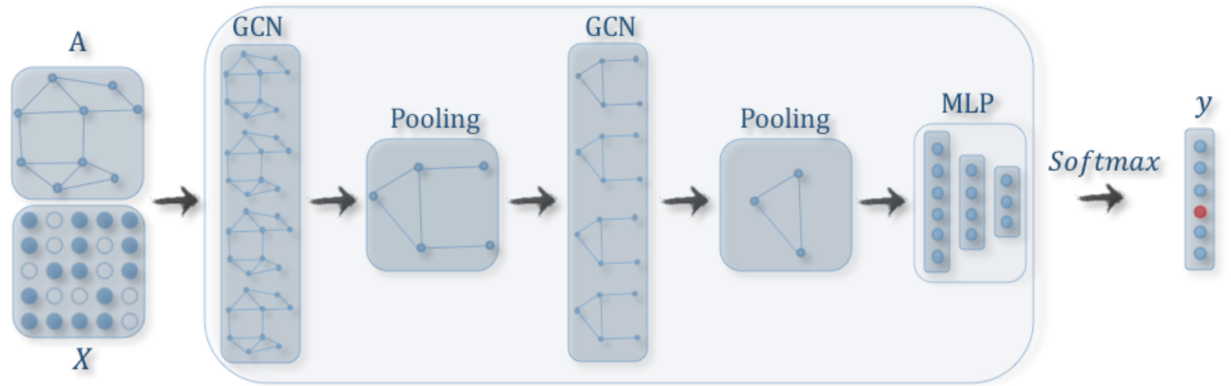

Figure 1: Graph Convolution Networks with Pooling Modules for Graph Classification

In graph classification problems,the complexity of graph data has introduced many challenges on current machine learning algorithms: graphs have different sizes of nodes and each node is related to a different number of neighbors, an interdependent relationship that captures complex information. A popular approach to do graph classification is to make use of graph kernels—functions which measure the similarity between graphs—plugged into a kernel machine,

such as a support vector machine. However, we can also adapt GCN to work on graph-level from node-level by interleaving graph pooling modules between GCN layers. As shown in Fig. 1, such an architecture design can be used to extract graph-level representations and to perform graph classification tasks. In our project, we followed similar architecture while feeding multi-scale features to the MLP layers. Our intuition is that while the final GCN layer reveals a high level structure of the graph, the previous GCN layers provide more detail about the graph. To utilize the rich information present in multi-scale features, we used $k-$max pooling as a graph coarsening tool. We have demonstrated the effectiveness of using multi-scale features in classification task for the PROTEINS and ENZYMES dataset.

2 Preliminaries

In this section, we provide definitions of basic graph concepts related to our project.

Graph We represent a graph G as (A, F), where $A \in \{0,1\}^{n\times n}$ is the adjacency matrix, and $F \in R^{n\times d}$ is the node feature matrix assuming each node has d features. Given a set of labeled graphs $D = \{(G_1, y_1), (G_2, y_2), ...\}$ where $y_i \in Y$ is the label corresponding to graph $G_i \in G$, the goal of graph classification is to learn a mapping $f : G \mapsto Y$ that maps graphs to the set of labels. The challenge — compared to standard supervised machine learning setup — is that we need a way to extract useful feature vectors from these input graphs. That is, in order to apply standard machine learning methods for classification, e.g., neural networks, we need a procedure to convert each graph to an finite dimensional vector in R^D.

Graph neural networks In this work, we build upon graph neural networks in order to learn useful representations for graph classification in an end-to-end fashion. In particular, we consider GNNs that employ the following general "message-passing" architecture:

$$H^k = M(A, H^{k-1}; \theta^k) \tag{1}$$

where $H^k \in R^{n\times d}$ are the node embeddings (i.e., "messages") computed after k steps of the GNN and M is the message propagation function, which depends on the adjacency matrix, trainable parameters θ^k , and the node embeddings H^{k-1} generated from the previous message-passing step. The input node embeddings H^0 at the initial message-passing iteration ($k = 1$), are initialized using the node features on the graph, $H^0 = F$.

There are many possible implementations of the propagation function M . For example, one popular variant of GNNs — Kipf's et al. [7] Graph Convolutional Networks (GCNs) — implements M using a combination of linear transformations and ReLU non-linearities:

$$H^k = M(A, H^{k-1}; \theta^k) = \text{ReLU}(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{k-1}W^{k-1}) \tag{2}$$

where $\tilde{A} = A + I, \tilde{D} = \sum_j \tilde{A_{ij}}$ and $W^k \in R^{d\times d}$ is a trainable weight matrix. In our project GNN module will run $K = 3$ iterations of Equation 1 to generate the output node embeddings $Z_k = H^k$ for $k = 1, \cdots, 3$. We feed these Z_k's to the pooling layer and then to the MLP layers for the final classification task.

3 Method

We have built a graph classifier that leverages node features at multiple scale to classify graphs. To derive a feature representation at a higher scale, we use GCN layer as a message propagation scheme between nodes to smooth the feature values of the nodes. We propagate message(i.e. feature values) multiple times to recompute the feature values of a node influenced by all the $k-$hop neighbors of that node. This technique reveals the structure of the graph by generating similar feature values within a strongly connected sub-graphs in the graph. We show this in Fig. 2, by using different colors for the raw feature values represented by the outer circle. After passing through convolutional layer, the colors start to show some structure of the graph (represented by the inner circles) as we can group the nodes by colors (blue, red, yellow, green).

Although the feature values at a higher scale reflect the structure of the network, they lose detail as a result of the smoothing process. In our work, we utilize the best of the both worlds by using feature values at multiple scales for the classification purpose (Fig. 3).

3.1 Network Architecture

We use graph convolutional layers introduced in [7] as the building blocks of our network. We stack three convolutional layer and concatenate the output of each layer which is then fed to a global pooling layer to generate a graph-level

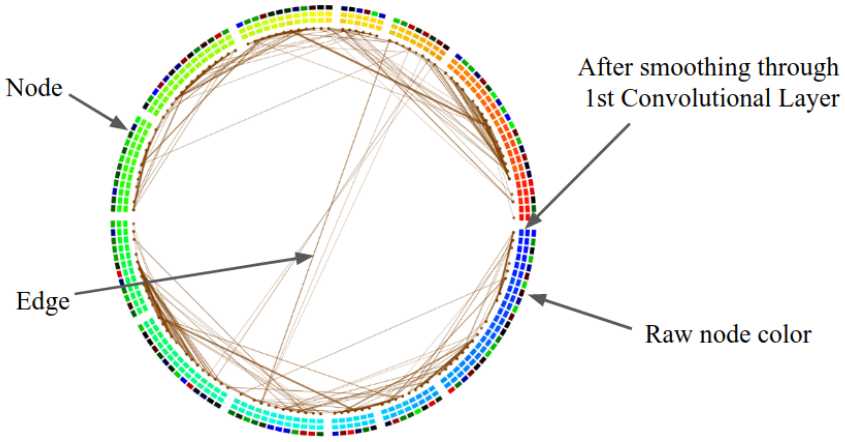

Figure 2: Node features at three different scales represent by layers of circle. Outer circle represents the raw node features which has a higher variation of colors. After message propagation though convolutional layers the feature values become smooth and reveal the higher order structure of the graph as demonstrated by the inner circles.

representation vector. This vector is passed through a dense layer to get the final prediction. We use cross-entropy loss as our loss function and train the network in mini-batches of size 32 to update the parameters for a iteration. We take the highest number of nodes for a graph in a mini-batch and zero pad all the other graphs to make them equal sized graphs.

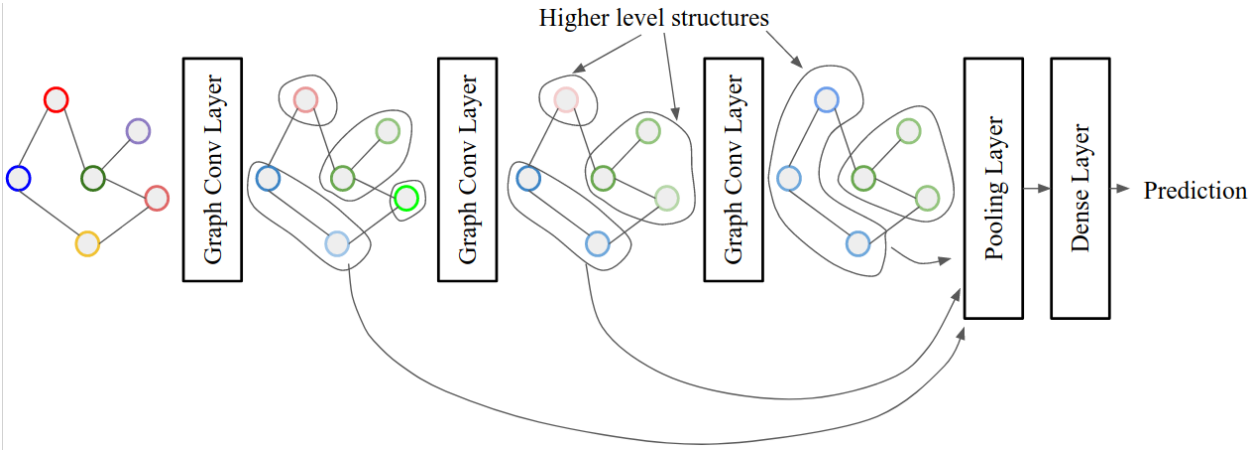

Figure 3: Network Architecture.

3.2 Pooling

To get the graph-level feature representation from the node-level features, we need to implement some sort of global pooling mechanism. In this project, we have used max-pooling and $k-$max pooling to get the final graph-level representation. In our experiment, we have found that max-pooling works better than average pooling and $k-$max pooling gives slightly better accuracy than max-pooling for $k = 5$ and $k = 10$.

4 Experiment

We have experimented with well-known PROTEINS and ENZYMES dataset. To demonstrate the effectiveness of the multi-scale features we run our experiment with just the last convolutional layer output instead of concatenating features from all layers. We refer this model as single-scale in contrast to multi-scale model. In Table 4, we see that multi-scale model outperforms single-scale model with $10-$max pooling but performs slightly worse with max-pooling and $5-$max pooling. We argue that max-pooling techniques aren't able to utilize the rich information present in multi-scale node features (Discussed in detail in 5).

Dataset	Scale	Max pooling	5-max pooling	10-max pooling
PROTEINS	Single-scale	74.92	76.37	75.65
	Multi-scale	74.47	76.36	**76.73**
ENZYMES	Single-scale	29.5	31.5	32.16
	Multi-scale	27.66	33.8	**36.1**

5 Discussion and Conclusion

In this project, we have explored the area of graph classification and particularly graph convolutional network. Our intuition is that we should see a graph at multi-scale rather than in single-scale to reveal the structure of the graph. We have computed multi-scale feature at node-level using graph convolutional layer. Extracting graph-level features from these node-level features is tricky. We have experimented with different max pooling techniques. We haven't seen any significant improvement using multi-scale feature over single-scale feature. We believe this is because the max-pooling technique is throwing away lot of the rich information present in the node-level features. This statement can be supported by the fact that when we increase the value of k, our multi-scale model begins to outperform the corresponding single-scale model. But increasing k also leads to overfitting which why we kept the value of k upto 10. Our next goal is to try some other pooling techniques such as sort-pooling [13] or diff-pooling [14] which can utilize multi-scale node features. We would also like to apply our method in a dataset from different domain like brain network where the modularity of the graph is higher.

References

[1] R. Girshick J. Redmon, S. Divvala and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, page 779–788, 2016.

[2] R. Girshick S. Ren, K. He and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[3] H. Pham M.-T. Luong and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, page 1412–1421, 2015.

[4] D. Yu G. E. Dahl A.-r. Mohamed N. Jaitly A. Senior V. Vanhoucke P. Nguyen T. N. Sainath et al G. Hinton, L. Deng. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. In *IEEE Signal processing magazine*, volume 29 of *6*, page 82–97, 2012.

[5] Y. Bengio et al Y. LeCun. Convolutional networks for images, speech, and time series. In *The handbook of brain theory and neural networks*, volume 3361 of *10*, page 1995, 1995.

[6] S. Hochreiter and J. Schmidhuber. Long short-term memory. In *Neural computation*, volume 9 of *8*, page 1735–1780, 1997.

[7] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations*, 2017.

[8] M. Bronstein F. Monti and X. Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, page 3697–3707, 2017.

[9] R. Ying W. L. Hamilton and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, page 1025–1035, 2017.

[10] B. Dai H. Dai and L. Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, page 2702–2711, 2016.

[11] R. Ying W. L. Hamilton and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, page 1025–1035, 2017.

[12] H. E. Sauceda S. Chmiela-A. Tkatchenko K. Schütt, P. J. Kindermans and K. R. Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems*, page 992–1002, 2017.

[13] M. Neumann M. Zhang, Z. Cui and Y. Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[14] C. Morris X. Ren W. Hamilton Z. Ying, J. You and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, page 4801–4811, 2018.