CS 6241 Final Project Report: Extending Graph Convolutional Networks to Edge Attributed Networks

Rohit Bandaru [*1]

Abstract

The graph convolutional network (GCN) (Kipf & Welling, 2016) is a powerful graph learning framework that has been successfully applied to different tasks. It is heavily inspired by convolutional neural networks, but with less expressivity by treating all of the nodes neighbors equally. This framework can be extended to use edge attributes to learn more complex representations of the adjacency matrix. I am proposing Edge and Node Attributed Graph Convolutional networks (EN-GCNs) to utilize more of the network data in the machine learning approach. This approach is applied to semi-supervised classification of the Cora citation dataset (Sen et al., 2008).

1. Introduction

Convolutional neural networks have achieved start of the art performance in many computer vision tasks. The intuition behind these 2D image convolutions is that pixels are related to nearby pixels and can be grouped into higher level features. There is also translation invariance in CNNs, because the same parameters are applied to all parts of the input. Only the pixel's neighbor's are considered in computing the output. CNNs and other standard neural networks are powerful and efficient models for learning, but they do rely on Euclidean data.

Graph convolutional networks (GCNs) (Kipf & Welling, 2016) are a method of applying neural networks to graph data, which can be non-Euclidean. They can be used for representation learning, community detection, or semi-supervised classification.

Graph convolutional networks impose less constraints on the input data compared to traditional CNNs. Convolutional neural networks require euclidean data. Images are rectangular and pixels are adjacent to the same number of pixels. GCNs however, have no constraint on the graph structure and can work on non-Euclidean data. Images can be interpreted as graphs, with pixels being adjacent to the pixels that are neighboring in the image. These neighbors are fixed and Euclidean. However, most network data lacks this structure and cannot be used with CNNs.

2. Graph Convolutional Networks

The standard graph convolutional network is formulated by its forward path (Kipf & Welling, 2016).

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)}) \tag{1}$$

A is the adjacency matrix. $H^{(l)}$ is the input feature matrix. $H^{(0)} = X$ is the input data of shape (N, D^0), where N is the number of nodes and D is the feature vector. X can be initialized as an identity matrix for node representation learning and community detection. σ is a nonlinear activation function such as the rectified linear unit (RELU). W^l is the weight matrix, which contains learnable parameters. As with a fully connected feed forward neural network, the input dimension of a layer's weights have to match the output dimension of the previous layer's weights.

In (Kipf & Welling, 2016) the adjacency matrix A is reformulated to add a self loop that would include the node's own features, and also normalize based on degree.

$$A = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} \tag{2}$$

The adjacency matrix is $\tilde{A} = A + I_N$. The identity matrix is added to introduce self loops so that the GCN layers considers the node's own features, in addition to its neighbors. The diagonal degree matrix $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is used to normalize the adjacency matrix. This is to prevent nodes with high degree from having significantly large outputs from GCN layers. This formulation essentially takes the average of a node and its neighbors when computing the output features in the forward pass for a node.

[*]Equal contribution [1]Computer Science, Cornell University, Ithaca, New York. Correspondence to: Rohit Bandaru <rb696@cornell.edu>.

3. Node Attributed Graphs

Many network datasets have attributes for each node, in addition to an adjacency graph. This is format that most of GCN research focuses on.

In (Li et al., 2018), community detection is done on networks with node attributes and structural embeddings.

4. Edge Attributed Graphs

To calculate the forward pass for a node, GCNs take the sum of the node and its neighbors. This is equivalent to having a kernel of all ones in CNNs. In CNNs the kernel parameters are trained and are more expressive.

Some datasets have attributes for each edge. This data can be passed through the neural network so that it does not treat each edge equivalently.

There is more limited research and datasets available for attributed graph learning or community detection.

5. Related Work

(Gong & Cheng, 2018) attempts to extend both GCNs and Graph Attention Networks (GAT) (Veličković et al., 2017) to utilize multidimensional edge features. In their approach, each edge feature is treated as a separate adjacency map. The outputs of the graph neural network layer operation are then concatenated for aggregation. My framework focuses on using learnable parameters on the edge features. This work is largely an extension of GAT to encompass GCNs and multidimensional edge features.

6. EN-GCN

There are multiple variations of my approach. The problem is formulated as follows. E is a matrix of shape (N, N, D_E) where D_E is the dimension of the edge attribute feature vector. $E_(ijk)$ is equal zero for all k if there is no edge between nodes i and j in the adjacency matrix. H remains the same as the input $X = H^0$. There is also the weight matrix W. EN-GCN appends additional weight matrices and computations to the GCN framework.

The goals of this approach as two-fold:

1. Incorporate high dimensional edge features in the GCN framework
2. Learn different and more robust representations of the adjacency matrix for each layer

Not only is the goal to use edge attributes within the GCN. The adjacency matrix should be propagated and dynamically changed for each layer. Different layers would be analyzing different patterns in the data and therefore interpret the spatial structure of the graph differently. Therefore, it would be intelligent to update the adjacency matrix for each layer. There are multiple approaches to implement these goals.

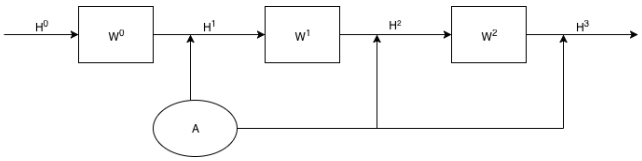

Figure 1. The standard graph convolutional network uses the same adjacency map for each layer

6.1. Independent Perceptrons for Weights

This approach seeks to learn a unique mapping from the edge features to 1 dimensional edge weights.

$$H^{(l+1)} = \sigma(EQ^{(l)}H^{(l)}W^{(l)}) \tag{3}$$

$Q^{(l)}$ is a weight matrix of shape $(D_E, 1)$. When multiplied with the (N, N, D_E) shaped edge feature matrix, it results in a (N, N) matrix that can be substituted into the GCN forward propagation in place of the adjacency matrix A.

This allows each layer to weight each feature differently in computing the edge weight. Different layers may focus on different edge features. One weakness of this approach is that it is only a one layer linear network or a perceptron. It cannot learn robust nonlinear representations of the edge features. This shortcoming is addressed in the other approaches. This approach will be called Independent Perceptrons for Weights (IPW).

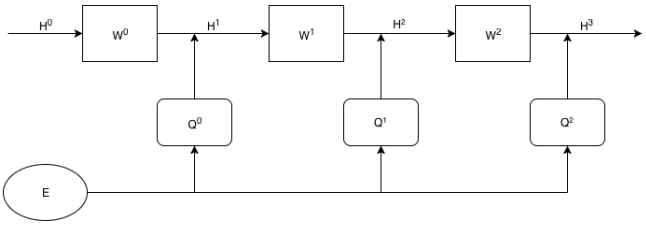

Figure 2. Three layer IPW network

6.2. Fully Connected Neural Network for Adjacency Matrix

This approaches seeks to have a forward propagation of the adjacency matrix, rather than use the same one in each layer. Both the node feature activations (H) and the edge feature activations (E) are propagated. Here we have edge feature propagations instead of using the same initial edge features

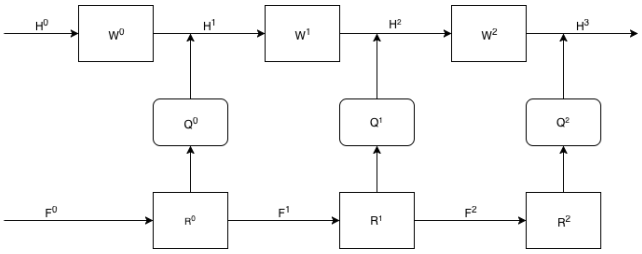

Figure 3. Three layer CPW network

matrix.

$$H^{(l+1)} = \sigma(F^{(l)}Q^{(l)}H^{(l)}W^{(l)}) \tag{4}$$

$$F^{(l)} = F^{(l-1)}R^{(l)} \tag{5}$$

F represents a propagated edge feature matrix. It is initialized to $F^0 = E$. $R^{(l)}$ is a weight matrix of shape (D_R^{l-1}, D_R^l). D_R^0 is equal to D_E. This is a neural feed-forward neural network on the edge feature matrix. It utilizes tensor multiplication as F is a three dimensional tensor. This approach will be called Connected Perceptrons for Weights (CPW).

6.3. Deep Regression of Weights

This approach replaces the single layer perceptron in IPW with a multi-layer perceptron. However, having different neural networks for every layer may lead to over-parametrization. The weights are shared for each layer.

This does not allow for unique representations of the adjacency matrix for each layer, but allows for learning on the edge feature attributes for a graph learning task. This approach will be called Deep Regression of Weights (DRW).

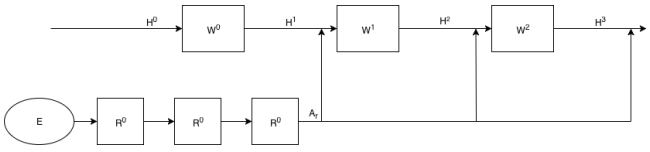

Figure 4. Three layer DRW network

7. Experiments

7.1. Implementation

PyTorch is being used as it allows for simple parameter learning and neural network configuration. The deep learning features such as parameter initialization and optimizers are also built in.

A challenge in the implementation was the edge feature matrix (E). It is large three dimensional matrix. For the Cora dataset it is of size (2708, 2708, 1433), which is over ten billion parameters. This cannot fit in memory, but the matrix is very sparse. PyTorch has weak support for sparse 3D matrices. To go around this, E is reshaped to be 2D and reshaped again after the tensor multiplication. PyTorch also does not support slicing sparse tensors. This requires constructing the sparse tensors by iterating through indices manually.

7.2. Reddit

The algorithm proposed can be applied to the Reddit Hyperlink Network and User and Subreddit Embeddings dataset (Kumar et al., 2018). The nodes represent different subreddits. The edges are directed and represent one subreddit having a hyperlink to another in a post. This dataset contains attributes for nodes and edges. The node attributes are embeddings based on which users post in the subreddit. The edge is attributed with a 80 features extracted from the text of the post containing the hyperlink. These are handcrafted features such as number of uppercase characters and VADER (Hutto & Gilbert, 2014) sentiment.

The dataset can contain multiple edges between the same pair of nodes. The approaches I have outlined are incompatible with this. As a pre-processing step, the mean of the feature vector is taken. Adding additional features, such as number of initial edges or standard deviation of features could be useful.

However, this dataset cannot be used for this project. It has node and edge attributes but no node labels to used for classification. It would be interesting to annotate the subreddits into different classes.

7.3. Cora

The Cora citation dataset (Sen et al., 2008) is a citation network dataset of papers in artificial intelligence. It contain 2708 papers. Each node represents a paper and has a feature vector of size 1433. This is a bag of words vector, where the values of either 0 or 1, representing whether a word is present in the paper. The nodes are classified into seven classes, which are subfields such as "genetic algorithms" and "theory". There are 5429 edges in the graph which represent citations. The dataset does not come with edge attributes. However, edge feature vectors can be simulated by taking the mean of the vectors. The edge feature vector would indicate which words are shared between the papers. Sharing certain key words can indicate that two papers are closely related.

7.4. Other Datasets

There are more limited options to find edge attributed network datasets. This approach can be demonstrated on ran-

dom or simulated edge and node attributed network datasets.

8. Results

The algorithm was evaluated using the Cora dataset. The data was split into training, validation, and testing sets. The same split as (Kipf & Welling, 2016)(140 training, 300 validation, and 1000 for testing) were used for comparison. The final results from the testing set are as follows.

GCN	IPW	CPW	DRW
81.0%	72.2%	62.2%	69.9%

The models were trained for 200 epochs, however CPW and DRW were stopped early after losses remained stable. The Adam optimizer (Kingma & Ba, 2014). The learning rate is set to 0.01. Weight decay is set to 5E-4. There is Dropout (Srivastava et al., 2014) of p = 0.5. Cross entropy loss is used for classification training. Rectified linear units are used as activation functions.

8.1. GCN

As a baseline, the GCN algorithm was implemented and tested alongside the three other architectures discussed in this report. The accuracy of (Kipf & Welling, 2016) is achieved. There are two layers with a hidden unit size of 100.

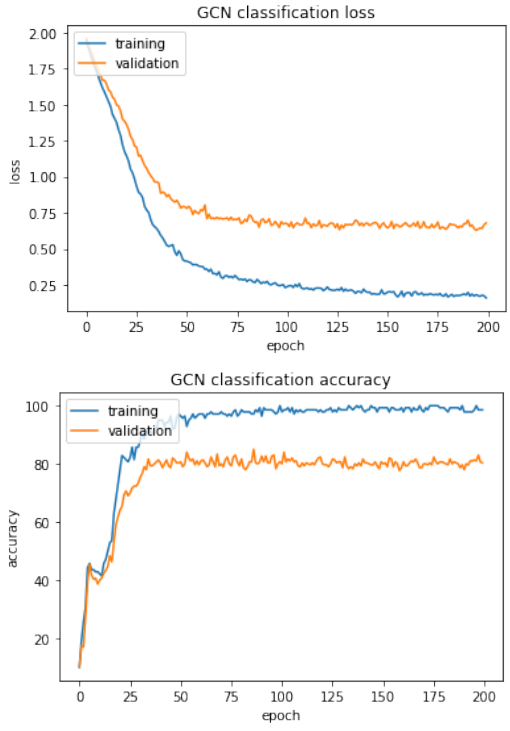

Figure 5. GCN Training

8.2. IPW

IPW replaces the adjacency matrix of GCN with a independently learned one for each layer. There is a learnable weight vector (Q) of size (d, 1) for each layer. These parameters are learned alongside the GCN weight parameters.

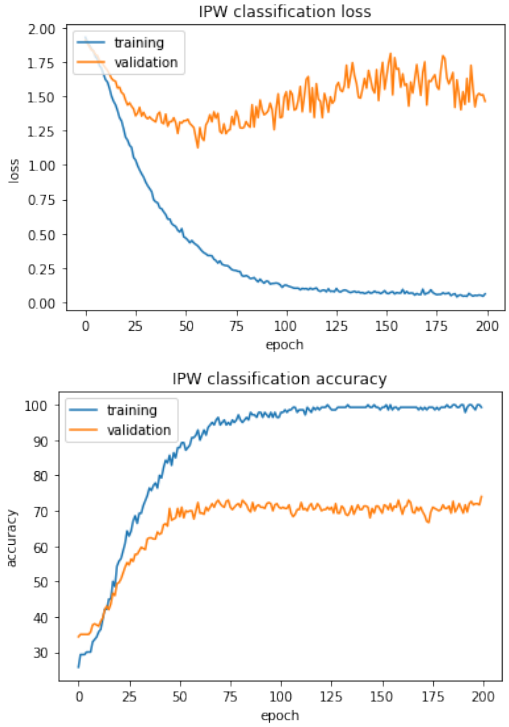

Figure 6. IPW Training

8.3. CPW

CPW adds a weight matrix (R) for each layer to update the edge feature matrix. This implementation uses a hidden unit size of 100.

8.4. DRW

DRW replaces Q and R with a multilayer network to learn an adjacency matrix from the edge feature matrix. For computational feasibility, this networks needs to be very small. It has two layers with a hidden unit size of 100.

9. Efficiency

One major drawback of applying a neural network on an edge feature matrix is that is computationally expensive. The time (seconds) to compute the forward pass on the Cora dataset for one epoch is shown:

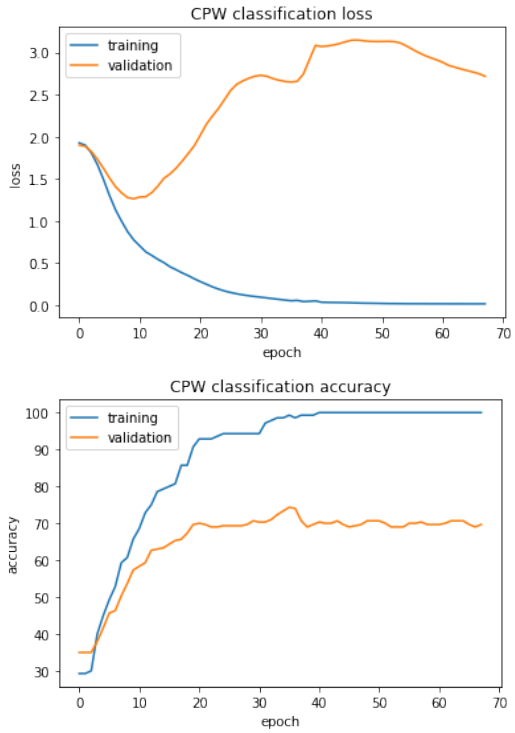

Figure 7. CPW Training

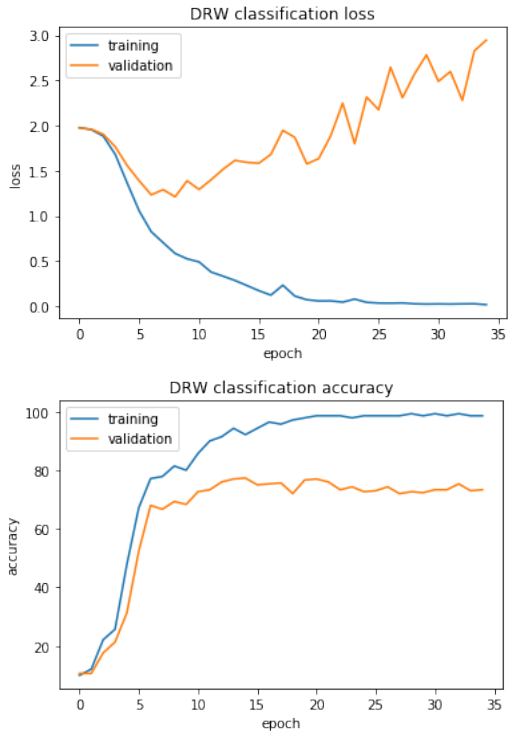

Figure 8. DRW Training

GCN	IPW	CPW	DRW
0.012 s	0.205 s	7.023 s	8.13 s

10. Discussion

The new GCN architectures proposed in this report fail to improve upon the performance of graph convolutional networks (Kipf & Welling, 2016) on the Cora dataset. However, these new approaches focus on leveraging edge attribute data. The edge attributes for the Cora dataset were derived from the node features. IPW, CPW, or DRW might outperform GCN on a dataset that has distinct edge and node attributes, as well as node labels for classification.

The problem shown in the experiments is overfitting. The accuracy follows the trend of number of parameters. GCN has the least parameters, so it overfits the least. While CPW has the most, and has the lowest accuracy. To further show this, a DRW network with only one R layer was tested. This would be adjacency matrix learning approach with the least number of parameters. This achieved a testing accuracy of 77.6%, which is above IPW, CPW, and DRW, but still below GCN. More research and experimentation is required to see how to address the problem. Traditional approaches such as increasing dropout and weight regularization were not effective. Perhaps, these methods would excel on larger and more complex datasets, as overfitting would be less likely.

This approach may also be useful for to learn node representations that include node attributes, edge attributes, and graph structure. This can be done on the Reddit dataset.

The code for this project is available at: `https://github.com/RohitBandaru/EN-GCN`.

References

Gong, Liyu and Cheng, Qiang. Adaptive edge features guided graph attention networks. *CoRR*, abs/1809.02709, 2018. URL `http://arxiv.org/abs/1809.02709`.

Hutto, Clayton J and Gilbert, Eric. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*, 2014.

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

Kipf, Thomas N. and Welling, Max. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

Kumar, Srijan, Hamilton, William L, Leskovec, Jure, and

Jurafsky, Dan. Community interaction and conflict on the web. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pp. 933–943. International World Wide Web Conferences Steering Committee, 2018.

Li, Ye, Sha, Chaofeng, Huang, Xin, and Zhang, Yanchun. Community detection in attributed graphs: an embedding approach. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Sen, Prithviraj, Namata, Galileo, Bilgic, Mustafa, Getoor, Lise, Galligher, Brian, and Eliassi-Rad, Tina. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

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