GRAPH CONVOLUTIONAL NETWORKS

CMU 11441/11741: MACHINE LEARNING WITH GRAPHS
Due date: 04/09/2024, 11:59 PM EST
https://github.com/cmu-ml4graph/gcn_assignment_s2024

Instructions

- **Allowed libraries:** This assignment involves implementing graph convolutional networks. You are **not** allowed to use any libraries that implement GCNs out of the box (like Pytorch-geometric). It is allowed to use autodiff libraries like Pytorch/Tensorflow. We highly recommend using Python + Pytorch for this assignment.
- **Getting feedback**: You can create a **private** fork of the repository on GitHub and add the TAs as collaborators (usernames: Edward-Sun). This might help you in asking questions without having to copy-paste your code on piazza (you can just reference Github code/copy permalink). You can use these instructions or just copy-paste the code into a new repository.
- **Posting your solutions online**: As with all the other assignments, please do not share your solutions publicly.

1 Introduction

Graph convolutional networks (GCNs) [Kipf and Welling, 2016] have emerged as the architecture of choice for learning meaningful graph representations. In this assignment, we will implement a GCN from scratch. We will then use the node representations learned by a GCN for three downstream tasks: i) node classification, ii) link prediction, and iii) graph classification.

Notations Let $\boldsymbol{G}(\boldsymbol{V},\boldsymbol{E})$ be a graph with node-set $\boldsymbol{V}$ and edge-set $\boldsymbol{E}$. Let $v \in \boldsymbol{V}$ be a vertex in the graph $\boldsymbol{G}$. The adjacency matrix of $\boldsymbol{G}$ is denoted by $\boldsymbol{A}$. $|\boldsymbol{V}|$ is used to denote the size of the vertex set.

2 GCN Review (30 points)

A GCN aims to learn node representations that can be useful for an end task. This is achieved by a message passing operation. Let $\boldsymbol{h}_v^0 \in \mathbb{R}^d$ be the initial node representation for a node $v \in \mathbb{V}$. This initial representation is then refined by running L-layers of a GCN [Kipf and Welling, 2016], where each layer $l+1$ is updated by using representations from the l^{th} layer as follows:

$$\boldsymbol{h}_v^{(l+1)} = \sigma\left(\frac{1}{|\boldsymbol{A}(v)|}\sum_{w\in\boldsymbol{A}(v)} \boldsymbol{W}^l\boldsymbol{h}_w^l + \boldsymbol{W}^l\boldsymbol{h}_v^l\right)$$
$$\boldsymbol{H} = [\boldsymbol{h}_0^L; \boldsymbol{h}_1^L; \ldots; \boldsymbol{h}_{|\boldsymbol{V}|-1}^L] \tag{1}$$

Where σ is a non-linear activation function, $\boldsymbol{W}^l \in \mathbb{R}^{k\times k}$ is the GCN weight matrix for the l^{th} layer, $\boldsymbol{A}(v)$ is the list of neighbors of a vertex v, and $\boldsymbol{H} \in \mathbb{R}^{|\boldsymbol{V}|\times k}$ is a matrix of the L^{th} layer representations the $|\boldsymbol{V}|$ nodes such that $\boldsymbol{H}_i = \boldsymbol{h}_i^L$. The matrix $\boldsymbol{H}$ is then passed to a downstream task, and the weight matrices $\boldsymbol{W}^l l \in [0, 1, \ldots, L]$ are learned during training.

Q1. What is the big-O time complexity of the computation expressed in Equation 1 in terms of $|\boldsymbol{V}|$, $|\boldsymbol{E}|$, d, k, and L? Your expression should not contain any other term. Assume $d < k$. $\mathcal{O}(|V| \times |E| \times k^2 \times L)$

Q2. What is the space complexity of the computation expressed in Equation 1 in terms of $|\boldsymbol{V}|$, $|\boldsymbol{E}|$, d, k, and L (assume intermediate terms are saved)? Your expression should not contain any other term.

3 Graph Exploration (20 points)

Our goal in this question is to understand representations used for implementing GCN and calculate some basic graph statistics.

We will use the notebook located in *notebooks/GraphExploration*. The notebook walks through the process of creating a graph object by loading

Please use *explore/explore_graph.py* notebook to instantiate a graph object (located in *data/graph.py*).

Each graph object has the following fields:

1. **features**: A tensor of shape $(|\boldsymbol{V}|, d)$, where the i^{th} element has the features for $v_i \in \boldsymbol{V}$.

2. **labels**: A tensor of shape $|\boldsymbol{V}|$, where the i^{th} element has the class label for v_i.

3. **adj**: A sparse adjacency matrix stored as a torch.sparse.coo matrix[1].

For the cora and citeseer graphs, please fill Table 3 with the required graph statistics using the graph object and the sparse adjacency matrix. To help you verify your solution, the numbers for the karate network have been added to the table (and are also present in the notebook).

[1]https://scipy-lectures.org/advanced/scipy_sparse/coo_matrix.html

Graph	Karate	Cora	Citeseer
Max in-degree	18	x	x
Min in-degree	2	x	x
Average in-degree	5.58	x	x
# nodes	34	x	x
# edges	190	x	x
Node feature dim	34	x	x

Table 1: Graph statistics

4 Implementing a GCN (100 points)

We will now implement a 2-layer GCN based on the equation outlined in Section 2. Please note again that the solution should not use any library except Pytorch/Python.

We will implement a GCN in two steps, outlined below.

1. Implement one layer of GCN by finishing the implementation of **GCNLayer** at *modeling/core/layers.py*. After you are done with your implementation, run the unit test at *tests/gcn_layer.py* to verify your solution on the small karate network. You can run the test with the command:

```
python -m unittest tests/test_gcn_layer.py
```

2. Finish the implementation of **GCN** at *modeling/core/gcn.py* by plugging in the **GCNLayer** implemented above. Please check your implementation by running the unit test:

```
python -m unittest tests/test_gcn_model.py
```

5 Node classification

5.1 Implementation (60 points)

We will utilize the GCN implemented in Section 4 for node classification. Please finish the implementation of **NodeClassifier** at *modeling/tasks/node_classification.sh*. Once you are done, run *scripts/run_node_classification.sh* with arguments *cora* and *citeseer* to report the following metrics on the test split (see the README for detailed usage):

In order to verify if your implementation is correct, you can try running it on the KARATE dataset, where you should get an accuracy of 1.0 (100%) and a loss of 0.

Graph	Accuracy %	Loss
KARATE	100	0
CORA	x	x
CITESEER	x	x

Table 2: Node classification results

5.2 Varying L (20 points)

For both CORA and CITESEER, modify the **GNN** to include $L = 3, 4, 5, 6$ layers and plot the loss and accuracy vs. L. Summarize your observations in 2-3 lines.

5.3 Topological features vs. inbuilt features (20 points)

In this sub-part, we will experiment with topological features proposed by [Benami et al., 2019]. For both CORA and CITESEER, you are provided with two additional node feature files:

1. *_topo*: graphs where nodes have just the topological features
2. *_plus_topo*: graphs with topological features added to the original features.

The files have already been provided for you to experiment with. For both CORA and CITESEER, report the performance obtained using the topological features (*_topo*) alone and in combination with the original features (*_plus_topo*). Is one of the methods better based on accuracy?

6 Link prediction

The goal of link prediction is to determine if an edge exists between two nodes $v_i, v_j \in \boldsymbol{V}$. We will build upon our GCN model and the node classifier to train a link predictor in this question.

6.1 Training data for link prediction (20 points)

A. The file *data/graph.py* includes a function to add edges to the training data by performing the negative sampling. Please use the notebook *notebooks/LinkPredictionTrainingData* to load CORA and CITESEER graphs and count the total number of positive and negative edges in the train split for each graph and report the counts in Table 3.

B. How is the training data for link prediction created? Please explain in 2-3 lines.

Graph	# Positive edges	# Negative edges
KARATE	190	190
CORA	x	x
CITESEER	x	x

Table 3: Training data statistic for link prediction

6.2 Implementation (80 points)

Finish the implementation of link prediction layer **LinkPrediction** in *link_prediction.py*

Once you are done, run *scripts/run_link_prediction.sh* with arguments 'cora' and 'citeseer' to report the loss and accuracy on the test split in Table 4. The results on KARATE network are already added to help you check your implementation. As with node classification, you can use the KARATE dataset to check your implementation. Please note that the accuracy and loss you observe might be off by ~ 2 points.

Graph	Accuracy %	Loss
KARATE	51.34	1.008
CORA	x	x
CITESEER	x	x

Table 4: Link Prediction Results

7 Graph classification

Our final task is graph classification. The goal of the graph classification task is to assign a label to *each graph* in the corpus. Note that the node classification task (Section 5) and the link prediction task (Section 6) operated with a single graph.

We will use two different graph classification datasets for this assignment: MUTAG (188 graphs, 2 classes) [Debnath et al., 1991] and ENZYMES [Borgwardt et al., 2005] (600 graphs, 6 classes). For the purposes of this assignment, the domain-specific details of these graphs are not important.

7.1 Graph Statistics (10 points)

Using the graphclassification notebook, load the graph datasets and calculate the average of the following statistics for the training split of the two graph datasets. Note that the dataset now has multiple graphs, thus these statistics need to be calculated over all graphs and then averaged. Please report these statistics in Table 5.

Graph	MUTAG	ENZYMES
Num graphs	141	x
Avg. num nodes	18.85	x
Avg. num edges	94.04	x
Node feature dim	8	x

Table 5: Graph statistics for the graph classification datasets

7.2 Implementation (90 points)

Typically, graph classification algorithms involve two steps:

1. Learning node representations: the embeddings for each node $v \in \boldsymbol{V}$ are obtained using a GCN.

2. The node embeddings are pooled into a single representation fed to a classifier (typically an MLP). The common choices for pooling involve mean-pooling (averaging) the node embeddings, max-pooling, and global-node pooling. The global-node pooling involves adding a global dummy node that is connected to all the nodes. The GCN representation of this global node is then fed to the classifier. For this assignment, we

Implement graph classification algorithm by completing *modeling/tasks/graph_classification.py*.

You can then run your code using *scripts/run_graph_classification.sh*. You can use the script to select the dataset and the pooling types. Please see the script for more details on the usage.

Please add the results in Table 6. To help you verify your implementation, the precision, recall, and F1 score for MUTAG using max-pooling has already been added to Table 6

Graph	MUTAG			ENZYMES		
	P	R	F1	P	R	F1
Mean-pooling	x	x		x	x	x
Max-pooling	84	83	83	x	x	x
Last-node pooling	x	x	x	x	x	x

Table 6: Graph classification results. Please use macro-averages to report the precision, recall, and F1 score for ENZYMES.

References

[Benami et al., 2019] Benami, I., Cohen, K., Nagar, O., and Louzoun, Y. (2019). Topological based classification of paper domains using graph convolutional networks. *arXiv preprint arXiv:1904.07787*.

[Borgwardt et al., 2005] Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56.

[Debnath et al., 1991] Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.