Implementing GNNs on MagmaDNN

Students: DINH Khanh Ly (City University of Hong Kong), Noah Dahle (University of Tennessee)
Mentors: Dr. Kwai Wong and Dr. Stan Tomov

Abstract

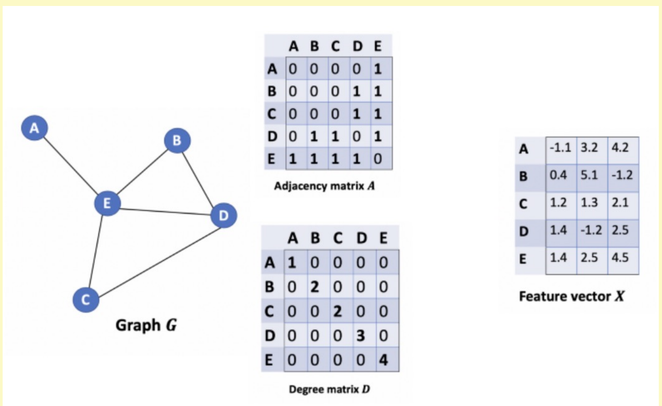
This paper presents two approaches to developing graph neural networks (GNNs) using MagmaDNN, a high-performance deep learning library designed for heterogeneous computing architectures. The approaches being implemented are the spectral and spatial methods. Our approaches leverage the parallel processing capabilities of MagmaDNN to enable efficient training and inference of GNNs on large-scale graphs. Our results demonstrate the potential of using MagmaDNN for developing high-performance GNN models for a wide range of applications in areas such as social network analysis, recommender systems, and drug discovery.

Introduction

- **Graph neural networks (GNNs)** have been successfully applied to a wide range of applications, including social network analysis, recommendation systems, drug discovery, and traffic prediction, among others. **MagmaDNN** is a high-performance deep learning library designed for heterogeneous computing architectures, such as multi-core CPUs, GPUs, and FPGAs.

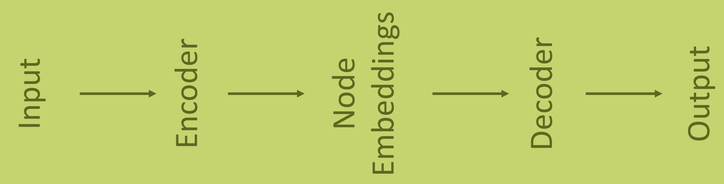
- There are many implementations of GNNs: attention, convolution, lstm, and more. The two models being developed in MagmaDNN are GCNs, specifically the **spatial** and the **spectral** methods. While the spatial method performs node representations aggregation using the mean rule, the spectral method transforms the graph signal into the spectral domain, performs convolution, then transforms it back into the graph domain.

- When performing these two methods, a graph, a feature matrix, an adjacency matrix, and a degree matrix are needed.

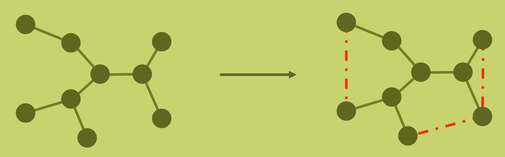


Implementation (Spatial GCN)

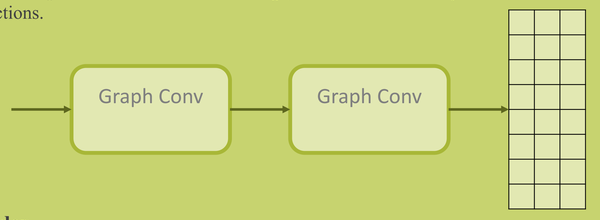
The Spatial GCN model uses a citation dataset named Cora, which consists of 2708 scientific publications and 5429 links. Each publication in the dataset is described by a dictionary that consists of 1433 unique words.



Input
The task of the model is binary classification, which decides whether a link exists (positive link) or does not exist (negative link) between two nodes. For that reason, the input is the original graph with added negative links.

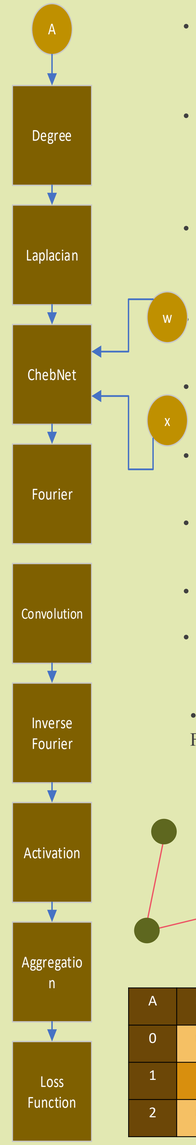


Encoder and Node Embeddings
The number of convolutional layers determines how far node representations are aggregated into the source node, therefore, it is limited by the depth of the graph. This simple model has two graph convolutional layer and an activation function between them. After processing the graph, the encoder produces new node embeddings, which is later used by the decoder to make predictions.

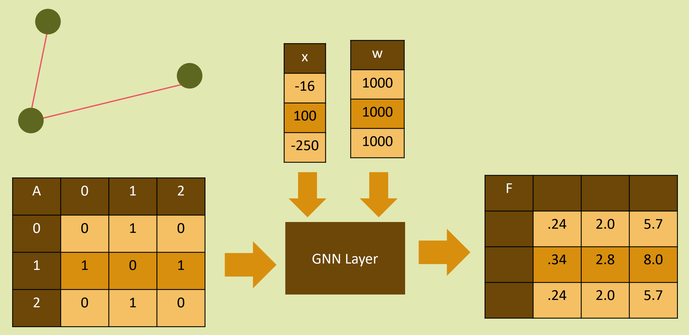


Decoder
Using the result of the encoder, the decoder computes a dot product of the node representations of two nodes on each edge. The new edge features is fed into an activation function to create a scalar score on every edge that shows the probability of edge existence.

Implementation (Spectral GCN)



- The spectral method of graph convolutional networks utilizes the convolution theorem by transferring the feature to the spectral domain, performing the convolution, and transforming it back into the spatial domain.
- This is a flowchart of the order of the spectral method. When given the adjacency matrix, feature vector, and arbitrary tensor of weights, this algorithm will perform the convolution and send its new feature vector into another layer or through to the loss function.
- The adjacency matrix goes through a series of transformations. The degree matrix is calculated by created a calculating the degree of each node and forming a diagonal matrix of those values.
- Then, the calculation $L = I - D^{-1/2}AD^{-1/2}$ is used to calculate the Laplacian matrix.
- From there, the Laplacian, feature, and weight matrices all get fed into the ChebNet function which calculates the filter matrix $g = \sum_{i=0}^{N} w[i]T_2(L)x$.
- After this, the filter and feature matrix undergo a graph Fourier transform and convolution: $X = U^T x \odot U^T g$.
- Then, the inverse Fourier is calculated to get back into the spatial domain: $X' = UX$
- Then, X' is sent through the RELU activation function : $\hat{X} = \text{RELU}(X')$.
- Finally, $\hat{X}$ is sent through the aggregation function, which updates each nodes features based on its neighbors: $F = \text{AGG}(\hat{X})$.
- The final equation for this model is $F = U[U^T x \odot U^T \sum_{i=0}^{N} w[i] * T_2(L)x]$



Future Work (Spatial GCN)

1. Implement the gradient function for the node pair multiplication operation to complete the decoder.
2. Add the sigmoid function to finish and start training the model.
3. Compare the result of the model to the GCN models made with PyTorch Geometric or DGL.

Future Work (Spectral GCN)

- Backpropagation is the next step. A rough gradient is written, but it is not quite ready.
- Collect a dataset to test the efficiency of this model in comparison to a mode made with PyTorch or Tensorflow.

Acknowledgements

Noah Dahle and DINH Khanh Ly would like to thank the National Science Foundation for their funding and support of the NICS RECSEM program.