TOPIC 17: GNN OPTIMIZATION

Qian Wang
wang432@usc.edu

Yi Deng
dengyi@usc.edu

Muhao Guo
muhaoguo@usc.edu

1 ABSTRACT

In this report, we briefly introduce what we have done for our GNN project by now. We first describe our problem, then the project timeline, which includes the papers we have read, the lecture nodes we have reviewed and the algorithms we have implemented. Then, we introduce the data set on which we use GNN optimization. Also, the algorithms we used are performed through pseudo-code and analyzed with flow chart. In the end, we sketched out our implementation and discussed the challenges we may come across.

2 PROBLEM DESCRIPTION

Graph study is trending now due to the ability of representing the real world such as social networks, bioinformatics, molecular structures, circuit elements etc. And nowadays GNNs has become a powerful and popular tool for machine learning on graphs. The purpose of this project is to implement GNN optimization based on PyTorch to solve problems in social networks. More specifically, our job can be split into two parts. For same-typed nodes, like citations, we use GCN to do node embedding. And for different-typed nodes, like users and movies, we apply PinSage on Netflix prize dataset in order to predict how the users rate movies.

3 PROJECT TIMELINE

Papers we read

- Graph Convolutional Neural Networks for Web-Scale Recommender Systems *by Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, Jure Leskovec*
- Advancing GraphSAGE with A Data-Driven Node Sampling *Jihun Oh, Kyunghyun Cho, Joan Bruna*
- Kernel Methods for Mining Instance Data in Ontologies *by Stephan Bloehdorn and York Sure*
- Modeling Relational Data with Graph Convolutional Networks *by Michael Schlichtkrull and Thomas N. Kipf*
- EDGNN: A Simple And Powerful GNN For Directed Labeled Graphs *by Guillaume Jaume and An-phi Nguyenz*
- Semi-supervised Classification with Graph Convolutional Networks *by Thomas N. Kipf and Max Welling*
- Graph Convolutional Neural Networks for Web-scale Recommender Systems *by Rex Ying, Ruining He, Kaifeng Chen, etc.*
- DeepWalk: Online Learning of Social Representations *by Bryan Perozzi, Rami Al-Rfou and Steven Skiena*

- node2vec: Scalable Feature Learning for Networks *by Aditya Grover and Jure Leskovec*
- Skipgram: Distributed Representations of Words and Phrases *by Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean*
- N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification *by Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi*
- A Comprehensive Survey on Graph Neural Networks *by Zonghan Wu, Shirui Pan, Fengwen Chen*
- Relational Inductive Biases, Deep Learning, and Graph Networks *by Battaglia, Peter W and Hamrick*

Github codes we have reviewed

- dgl
- Deepwalk
- Node2vec

Lecture Notes we have went through

- Stanford CS224W: Graph Neural Networks
- Stanford CS224W: Graph Neural Networks: Hands-on Session
- Stanford CS224W: Message Passing and Node Classification
- Stanford CS224W: Machine Learning with Graphs (Video)

Expect to do

- Keep updating the Github repo we created for GNN optimization on Netflix prize dataset.
- Train the model on entire Netflix Prize dataset.
- Make a prediction on how the users rate the movies.

4 ANALYSIS OF GCN

- **Algorithm**
  The graph convolutional operator(GCN) from the "Semi-supervised Classification with Graph Convolutional Networks" paper
  A multi-layer Graph Convolutional Network (GCN) with the following layer-wise propagation rule:
  $H^{(l+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)})$

  $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$denotes an activation function, such as the $ReLU(\cdot) = max(0, \cdot)$. $H^{(l)} \in R^{N \times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$.

- **Implementations**
  The implementations are based on PyTorch.We used a GCN model to solve node classification problem.

  Data and Structure:

dataset	num of edges	num of nodes	size of node features	num of num_classes
cora	126842	19793	8710	70
cora_ml	16316	2995	2879	7
citeseer	10674	4230	602	6
dblp	105734	17716	1639	4
pubmed	88648	19717	500	3

Figure 1: Datasets of GNNstack

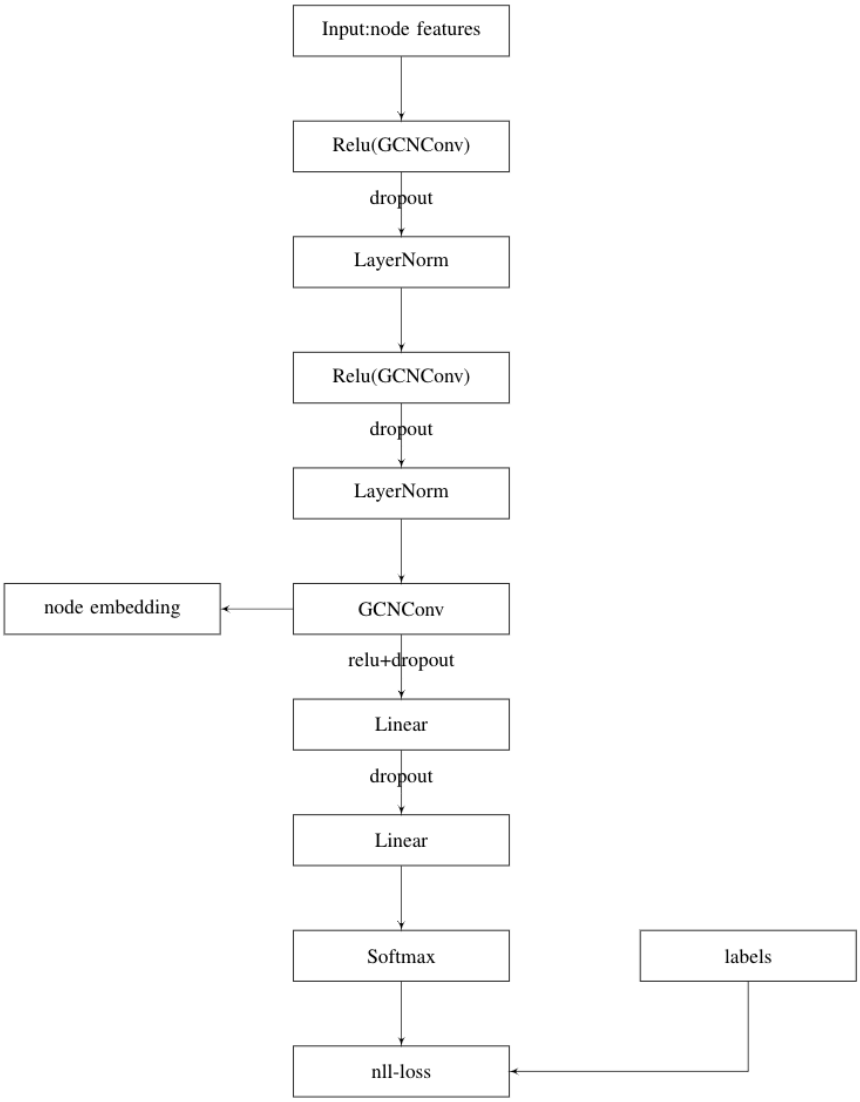

Figure 2: GNN stack model structure

Results:

dataset	cora	cora-ml	citeseer	dblp	pubmed
accuracy	0.8480	0.9790	0.9960	0.9434	0.9328

5 ANALYSIS OF PINSAGE

- **Algorithm**
  Left: A small example input graph. Right: The 2-layer neural network that computes the embedding hA2 of node A using the previous-layer representation,hA1, of node A and that of its neighborhood N (A) (nodes B,C,D)

  The core of our PinSage algorithm is a localized convolution operation, where we learn how to aggregate information from u's neighborhood.

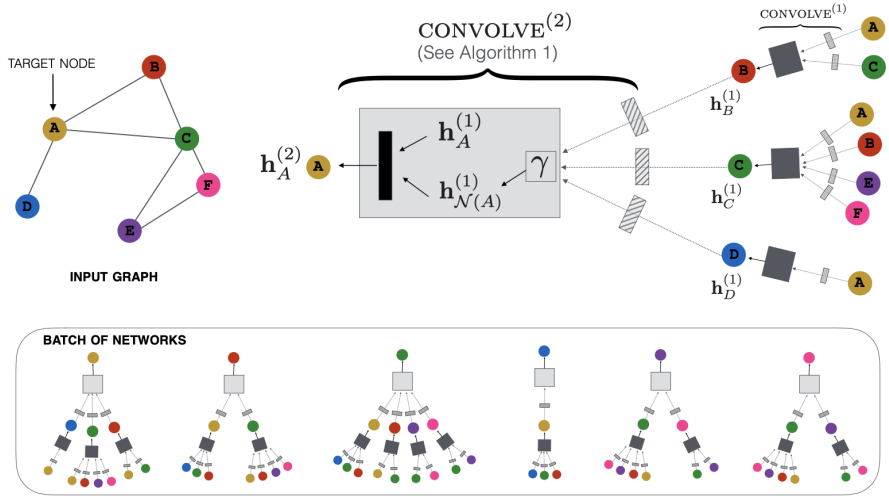

Figure 3: PinSage

Algorithm 1: CONVOLVE

Input : Current embedding $\mathbf{z}_u$ for node u; set of neighbor embeddings $\{\mathbf{z}_v | v \in \mathcal{N}(u)\}$, set of neighbor weights $\boldsymbol{\alpha}$; symmetric vector function $\gamma(\cdot)$

Output: New embedding $\mathbf{z}_u^{\text{NEW}}$ for node u

$\mathbf{n}_u \leftarrow \gamma\left(\{\text{ReLU}\left(\mathbf{Q}\mathbf{h}_v + \mathbf{q}\right) \mid v \in \mathcal{N}(u)\}, \boldsymbol{\alpha}\right)$;

$\mathbf{z}_u^{\text{NEW}} \leftarrow \text{ReLU}\left(\mathbf{W} \cdot \text{CONCAT}(\mathbf{z}_u, \mathbf{n}_u) + \mathbf{w}\right)$;

$\mathbf{z}_u^{\text{NEW}} \leftarrow \mathbf{z}_u^{\text{NEW}} / \|\mathbf{z}_u^{\text{NEW}}\|_2$

Figure 4: PinSage convolve

Algorithm 2: MINIBATCH

Input : Set of nodes $\mathcal{M} \subset \mathcal{V}$; depth parameter K;
neighborhood function $\mathcal{N} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$

Output: Embeddings $\mathbf{z}_u, \forall u \in \mathcal{M}$

/* Sampling neighborhoods of minibatch nodes. */

$\mathcal{S}^{(K)} \leftarrow \mathcal{M}$;

for $k = K, ..., 1$ **do**

  $\mathcal{S}^{(k-1)} \leftarrow \mathcal{S}^{(k)}$;

  for $u \in S^{(k)}$ **do**

    $\mathcal{S}^{(k-1)} \leftarrow \mathcal{S}^{(k-1)} \cup \mathcal{N}(u)$;

  end

end

/* Generating embeddings */

$\mathbf{h}_u^{(0)} \leftarrow \mathbf{x}_u, \forall u \in \mathcal{S}^{(0)}$;

for $k = 1, ..., K$ **do**

  for $u \in \mathcal{S}^{(k)}$ **do**

    $\mathcal{H} \leftarrow \left\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\right\}$;

    $\mathbf{h}_u^{(k)} \leftarrow \text{CONVOLVE}^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathcal{H}\right)$

  end

end

for $u \in \mathcal{M}$ **do**

  $\mathbf{z}_u \leftarrow \mathbf{G}_2 \cdot \text{ReLU}\left(\mathbf{G}_1 \mathbf{h}_u^{(K)} + \mathbf{g}\right)$

end

Figure 5: PinSage minibatch

Algorithm 1 Basic Random Walk; q is the query pin; E denotes the edges of graph G; α determines the length of walks; N is the total number of steps of the walk; V stores pin visit counts.

BasicRandomWalk(q: Query pin, E: Set of edges, α: Real, N: Int)

totSteps = 0, $V = \vec{0}$
repeat
  currPin = q
  currSteps = SampleWalkLength(α)
  for i = [1 : currSteps] **do**
    currBoard = E(currPin)[rand()]
    currPin = E(currBoard)[randNeighbor()]
    V[currPin]++
  totSteps += currSteps
until totSteps $\geq N$
return V

Figure 6: random walk

- **Data and Structure:**
  Data
  source from : http://www.netflixprize.com The movie rating files contain over 100 million ratings from 480 thousand randomly-chosen, anonymous Netflix customers over 17 thousand movie titles. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received during this period. We use data from Netflix, there are total 17770 movies and CustomerIDs range from 1 to 2649429, with gaps.
  Structure

- **Result:**

6 TOOL (SOFTWARE) CHALLENGES

- We mainly used PyTorch.
- Besides PyTorch, we will use Python and we work on Github, Linux and Overleaf.
- We create a virtual environment of Python.
- We use DEEP GRAPH LIBRARY (DGL)

7 PROPOSAL

Introduction
Most of the current deep learning processing data is limited to images and text, but it does not include the network.This is because the network is the connection information between nodes and it is difficult to convert this kind of information to standard data. How to solve this problem? If we can convert the network to a geometric space, in other words, assignment to each node a coordinates in a geometric space, then we can use the general machine learning methods to solve this kind of problem. So we use DeepWalk or Node2vec algorithms to give every node in the network a vector representation. And for further specific task like node classification, we can also use GCN algorihtm to train the model.GCN is usually used in the network with same-typed nodes, for different-typed nodes, like users and items, we choose Pinsage Algorithm to train the model.

prior work summary

- We have learned how to use Pytorch.
- We have read the lecture notes from Stanford University about Machine Learning with Graphs.
- Implemented node2vec and deepwalk in pytorch.
- We have get some extra data sets (like BolgCatalog and Homo Sapiens etc.) and implement the node2vec data sets.
- Implemented GCN in pytorch and trained GNNstack model for node classification.

the key novelties of your work

- We have implement the whole GNN algorithm on some data sets
- Use data sets that are closely related to life.
- Perform a new machine learning method on the network.
- Use Pinsage Algorithm on recommandation and rating problem.