Coding GNNs

GDL 2022
Group 13
Project 01

Vinaik Chhetri, Marc Salvadó Benasco

Abstract
This project aims to implement at least 3 architectures and recreate the results in Spektral. This paper discusses models in the following papers: "How Attentive are Graph Attention Networks?", "Beyond Low-frequency Information in Graph Convolutional Networks" and "Simple and Deep Graph Convolutional Networks". All three models were successfully implemented/ported and they were able to achieve similar results to the ones in the paper.

1 Introduction

A great amount of data can be modelled as a graph and GNNs (Graph Neural Networks) have been widely used to deal with such data. Spektral and PyG are Python libraries to train GNNs; the first one, based on Keras (TensorFlow), and the second one, on PyTorch.

Majority of the GNNs models are mainly implemented using PyG whereas Spektral lacks sufficient architectures/layers. Our task is then to implement some of these models in Spektral and recreate the results published in the papers. Two of the models that were implemented were only trained on synthetic datasets to see the core idea/principles behind the model. Successfully obtaining similar/same results on the synthetic datasets should guarantee that it would perform similarly on the other datasets the author experiemented on.

The three models that were chosen were:

1. GATv2 ("How Attentive are Graph Attention Networks?") [1]
2. FAGCN ("Beyond Low-frequency Information in Graph Convolutional Networks") [2]
3. GCNII ("Simple and Deep Graph Convolutional Networks") [3].

- GATv2

GATv2 is a natural extension of GAT(Graph Attention Model) which was engineered to overcome the static nature of GAT's attention. Shaked Brody et.al. modify the existing GAT architecture to generate dynamic attention. To demonstrate the advantage of GATv2 over GAT, they propose/train on a synthetic dataset("DICTIONARYLOOKUP") and other datasets. We simply tried to only recreate the results for the synthetic dataset as this dataset clearly showed GATv2's superiority over GATv. The synthetic dataset generator has a parameter k which constructs a kxk bipartite graph dataset of cardinality $k!$. Shaked Brody et.al. trained 4 models over $k = 4...20$. We were successfully able to reconstruct the results but we could only train the models over $k = 4, 5, 6$ due to high computation time.

- FAGCN

According to Deyu Boet.al FAGCN is a generalization of most existing GNNs([2]). Most GNNs only make use of low frequency signals of node features. The low pass filters in GNNs are only good for assortative networks that always have similar nodes features connected making it unsuitable for inference on disassortative networks([2]). FAGCN was thus created to incorporate both types of signals during aggregation to deal with real world graphs that have a mix of assortative and disassortative edges/graphs. To demonstrate the advantage of FAGCN over low frequency filters and high frequency filters, Deyu Boet.al proposes/trains on a synthetic dataset which has 2 classes of nodes with a parameter p and q to control the assortativity/disassortativity of the network. We were able to successfully produce their results using their dataset.

- GCNII

GCNII is an extension of the well-known GCNs (Graph Convolutional Networks), which are a generalisation of CNN (Convolutional Neural Networks) for graph-structured data. GCNs are known to suffer from **oversmoothing**, which is the fact that, as the number of layers with non-linearities grows (increasing the depth of the model), the nodes outputs tend to be indistinguishable. This is why deep GCN perform worse than shallow GCN, which shallowness has the limitation of not being able to approximate the high-frequency components of the data. The goal of GCNII is then to provide more depth to GCN while avoiding the oversmoothing problem.

In section 2 we will introduce the main idea of the aforementioned papers; in section 3, we will explain the used methodology; in section 4, we will explain the steps that have been done during the implementation, as well as the main challenges and problems; in section 5, we will present a comparison of the results between our implementations and the existing implementations in PyG; and in section 6 we will discuss the results and conclude the paper, (as well as suggest future works?).

2 Models

- GATv2

As mentioned above GATv2 is a modification of GAT with an enhanced attention mechanism. GAT, unlike other vanilla GNNs does not use an aggregation function that equally weighs all neighbours of a node but rather learns a vector of weights via a single layered neural network. This can be seen as follows:

$$\begin{aligned} e(h_i, h_j) &= \sigma_{lr}(a^T[Wh_i||Wh_j]) \\ \alpha_{ij} &= \frac{exp(e(h_i, h_j))}{\sum_{j' \in N_i} exp(e(h_i, h_{j'}))} \\ h'_i &= \sigma(\sum_{j \in N_i} \alpha_{ij} W h_j) \end{aligned} \tag{1}$$

Note: σ_{lr} is LeakyReLU()
According to the author the computation of the unnormalized scores $e(h_i, h_j)$ creates static attention——Given a family of scoring functions $F \subseteq (\mathcal{R}^d X \mathcal{R}^d \mapsto \mathcal{R})$ for a set of keys $\mathcal{K} = k_1, k_2, ...k_n \subset \mathcal{R}^d$ and for a set of queries $\mathcal{Q} = q_1, q_2, ...q_m \subset \mathcal{R}^d$, all functions $f \in F$ computes static attention if for each f there exists $j_f \in [n]$ such that for each query $i \in [m]$ and each key $j \in [n]$, $f(q_i, k_{j_f}) \geq f(q_i, k_j)$. In short no matter which function is chosen from a family of function that computes static attention, it will always have a key that scores highest regardless of the query. The author claims that the unnormalized scores $e(h_i, h_j)$ has both the parameters inside the leaky relu function thus constraining its degrees of freedom. Due to the monotonicity of leaky relu and softmax, $e(h_i, h_j)$ and the attention weights α_{ij} will always be maximum for maximum $h*$ among all hidden node representation thus generating static attention.

This problem inspired the authors to create GATv2 which computes dynamic attention——Given a family of scoring functions $F \subseteq (\mathcal{R}^d X \mathcal{R}^d \mapsto \mathcal{R})$ for a set of keys $\mathcal{K} = k_1, k_2, ...k_n \subset \mathcal{R}^d$ and for a set of queries $\mathcal{Q} = q_1, q_2, ...q_m \subset \mathcal{R}^d$, all functions $f \in F$ computes dynamic attention if for any mapping $\phi : [m] \mapsto [n]$ there exists $f \in \mathcal{F}$ such that for each query $i \in [m]$ and each key $j \neq \phi(i)$, $f(q_i, k_{\phi(i)}) \geq f(q_i, k_j)$. In shorts the maximal key is query dependent meaning that there exists a different key association for a different query. GATv2 fixes the problem by taking the weight vector a, seen in equation(1), outside of the LeakyRelu activation function thus giving us:

$$e(h_i, h_j) = a^T \sigma_{lr}(W[h_i||h_j]) \tag{2}$$

- FAGCN

The architecture of FAGCN involves first applying a multi-layered perceptron layer on the node features which is then sent through the adaptive frequency aggregation layer $L - 1$ times. Then a final affine transformation is applied to the output. The model can be seen as the following:

$$\begin{aligned} h_i^{(0)} &= \phi(W_1 h_i) \\ h_i^{(l)} &= \epsilon h_i^{(0)} + \sum_{j \in N_i} \frac{\alpha_{i,j}^G}{\sqrt{d_i d_j}} h_j^{(l-1)} \\ h_{out} &= W_2 h_i^{(L)} \\ \alpha_{ij}^G &= \tanh\big(g^T[h_i||h_j]\big) \end{aligned} \tag{3}$$

Note: $\alpha_{ij}^G = \alpha_{ij}^L - \alpha_{ij}^H$ where α_{ij}^L is the proportion of node j's low frequency signal to node i and α_{ij}^H is the proportion of node j's high frequency signal to node i. Since α_{ij}^G is an output of a tanh function, $\alpha_{ij}^L + \alpha_{ij}^H = 1$. If $\alpha_{ij}^G > 0$ then the low frequency signal from node j to i is greater than the high frequency signal from node j to i. g is a shared convolutional kernel between the connecting nodes. $\epsilon \in [0, 1]$.

- GCNII

Given a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, $|\mathbb{V}| = n$, $|\mathbb{E}| = m$; let's consider the same graph with a self loop per node, $\widetilde{\mathbb{G}} = (\mathbb{V}, \widetilde{\mathbb{E}})$, with adjacency matrix $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ and degree matrix $\widetilde{\boldsymbol{D}} = \boldsymbol{D} + \boldsymbol{I}$, where $\boldsymbol{A}$ and $\boldsymbol{D}$ are the adjacency and degree matrices of $\mathbb{G}$.

Given the node features $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, which dimension can be downsized via dense layers to $\boldsymbol{H} \in \mathbb{R}^{n \times \widetilde{d}}$, $\widetilde{d} < d$, a GCN layer is defined as $\boldsymbol{H}^{(l+1)} = \sigma(\widetilde{\boldsymbol{P}} \boldsymbol{H}^{(l)} \boldsymbol{W}^{(l)})$, where $\boldsymbol{H}$ are the hidden layers, $\boldsymbol{W}$ are the weight matrices, σ stands for ReLU and $\widetilde{\boldsymbol{P}} = \widetilde{\boldsymbol{D}}^{-1/2} \cdot \widetilde{\boldsymbol{A}} \cdot \widetilde{\boldsymbol{D}}^{-1/2}$.

The GCNII layer suggests two improvements: <u>initial residual connection</u> (first term of the product inside the sigma), in order to ensure that each node output contains a fraction of the first hidden layer, despite the high number of convolutonial layers; and <u>identity mapping</u> (second term of the product) —following the philosophy of ResNets, which have proven to work and have properties like having a small norm for $\boldsymbol{W}$—. Thus, the GCNII layer is defined as

$$\boldsymbol{H}^{(l+1)} = \sigma\Bigg(\Big((1-\alpha_l)\widetilde{\boldsymbol{P}}\boldsymbol{H}^{(l)} + \alpha_l \boldsymbol{H}^{(0)}\Big) \cdot \Big((1-\beta_l)\boldsymbol{I}_n + \beta_l \boldsymbol{W}^{(l)}\Big)\Bigg)$$

3 Methodology

The authors for GATv2 and FAGCN trained their models on multiple datasets but they both had synthetic datasets which demonstrated the core principle and idea behind their model so we only trained GATv2 and FAGCN on these datasets and recreated only results pertaining to them. And secondly due to time/experience and computational power constraints, we were not able to train the models on all the datasets. But we were able to train our third model GCNII on multiple datasets.

- GATv2

For this part of the project both GATv2 and GAT were implemented and trained on the synthetic dataset "DICTIONARY-LOOKUP. The datset is thoroughly explained in Section 4.1.. Firstly the models were only trained for datasets of $4!4 \times 4$ and $5!5 \times 5$ bipartite graphs mainly due to inefficient training measures and also due to low computational power. Shaked Brody et.al have implemented a class that returns training and validation dataset of $k \times k$ bipartite graphs. They implemented this using PyG and we made modifications to it using Spektral. They trained a single-headed and eight-headed GAT and GATv2 on the "DICTIONARYLOOKUP" dataset for $k = 1, 2, ...20$. They were able to show that a single-headed GATv2 outperforms an 8-headed and 1-headed GAT on training and test datasets. The only code that we used from the authors was the data generator. We instead used the implementation of GAT in Spektral as a starter. We implemented 2 GATv2 models. One GATv2 model specifically deals with the synthetic dataset containing direct/asymmetric edges whilst the other GATv2 model deals with datasets containing symmetric edges. Our model only deals with a Single Loader meaning our model only takes in 1 graph at a time. This is clearly not optimal for training datasets with many graphs. Our model was implemented well but our training procedure is not optimal—the ideal situation would have been to use batch gradient descent but we instead averaged our gradients to make an update to the weights. This was done due to time constraints and lack of experience with Disjoint/Batch loaders and Spektral in general.

- FAGCN

The authors of FAGCN also used a synthetic dataset which is thoroughly described in section 4.1. They also implemented a data generator using numpy and SciPy which made it convenient for us to use. The authors created 10 datasets each containing 200 nodes from 2 different classes and the datasets were created from low disassortativity to high disassorativity. A dataset of 200 nodes were split into 2—training and test dataset. These datasets were then fed into FAGCN to experiment with high, low and adaptive frequency filters. According to Deyu Boet.al FAGCN can control the proportion of low/high frequency signals to node i from j via α_{ij}^{G} as shown in equation(3). The authors did not explicitly state how they trained on the dataset using high frequency and low frequency filters to produce Figure 1.a. in the paper "Beyond Low-frequency Information in Graph Convolutional Networks" [2]. So we found it plausible to use $\alpha_{ij}^{G} > 0$ as a low frequency retainer/filter and $\alpha_{ij}^{G} < 0$ as a high frequency retainer/filter. Whereas α_{ij}^{G} in FAGCN is not strictly positive or negative as FAGCN has the property of an adaptive frequency filter. Hence inorder to mimic a low frequency filter the α_{ij} matrix in our code is put in absolute to make all its entries positive. And inorder to mimic a high frequency filter the α_{ij} matrix in our code is put in absolute value and multiplied by -1 to make all its entries negative. Wheras for FAGCN we leave it as it is. This approach of taking absolute value and negating does not purely retain high frequency signals and only taking absolute value of the α_{ij} matrix does not purely retain low frequency signals either. So a better alternative would be to set α_{ij} matrix to a matrix of ones for a pure low frequency signal filter and set α_{ij} matrix to a matrix of -1 for a pure high frequency signal filter. $1 = \alpha_{ij}^{L} + \alpha_{ij}^{H}$ so $\alpha_{ij}^{L} = 1$ would imply $\alpha_{ij}^{G} = 1$ and $\alpha_{ij}^{H} = 1$ would imply $\alpha_{ij}^{G} = -1$. We were able to train FAGCN properly without a custom training loop. It was easy to train because it did not require us to train the data as a batch but as a single graph instead.

- GCNII

For the implementation in TensorFlow and Spektral, we took the original code [4] (in PyG) from the authors of the paper. The files that were modified were `train.py`, `model.py` and `utils.py`. The main challenge was to learn how to adapt the functions and operations from PyTorch to TensorFlow and Spektral; some of them couldn't be ported from one library to another, but they had to be rewritten in a different way, like the dataloaders and the training process.

A problem we encountered was that we couldn't apply the AdamW (Adam Weight Decay) optimizer; we tried using the function AdamW from TensorFlow-addons, but it was impossible to set 2 different weight decays for the fully-connected layers on the one side and for the graph convolution layers on the other side, as is done in the original code. This is likely to provoke a divergence in the results.

4 Implementation

4.1 Datasets

- GATv2

Authors of GATv2 proposes a synthetic dataset (DICTIONARYLOOKUP). They build a data generator that accepts parameter k and builds a k! kxk bipartite graph with 2 attributes each for all nodes. The first 1,...,k nodes are the query nodes and the second set of k nodes k+1...2k are key nodes. The 2nd attribute for the nodes are values whereas as the 1st attribute are the name/identifier of the node. For the first set of nodes 1...k, the second attributes are unknown. Based on its 1st attribute and edge connections it needs to determine its value attribute.

- FAGCN

The authors propose a synthetic dataset and they build a generator for it. It takes in parameters p and q. p determines the probability for a node in a class to connect to a node in the same class whereas q determines the probability for a node in a class to connect to a node in another class. p is fixed to 0.05 and q ranges from 0.01 to 0.1 incrementing by 0.01. The purpose of the generator is to create dataset that go from assortative to disassortative networks.

- GCNII

The original code provides 12 different 1-graph-datasets, some of them are very large and take a lot of time to train. We have selected 4 of them, called "pubmed", "cornell", "texas" and "wisconsin". "Pubmed" is a benchmark dataset about citations, taken from [5], where nodes are documents; edges, citations; and node features, a bag-of-words representation of the document. The other three datasets are web networks, where nodes are web pages; edges, hyperlinks; and node features, again, a bag-of-words representation of the page; and which are taken from [6].

The number of nodes of the datasets are, respectively: 19717, 183, 183 and 251; the number of edges, respectively: 44338, 295, 309 and 499; the dimension of the node features, respectively: 500, 1703, 1703 and 1703; and their number of classes, respectively: 3, 5, 5 and 5.

4.2 Hyperparameters

- GATv2

Dropout was set to 0 to see GAT and GATv2's ability to generalize. Learning rate was set to 0.001 as the authors chose it as default. Experiments were conducted using 8 heads and 1 heads.

- FAGCN

$\epsilon = 0.3$ was chosen because author had set 0.3 as default. Dropout was set to 0 to only see the the effect of different frequency filters without any supervision/tweaking. learning rate was set to 0.01 as the authors chose it as default.

- GCNII

As mentioned above, we couldn't use the Adam with weight decay as in the original code, so we used raw Adam. To find the best learning rate, we tried the values of 0.5, 0.1, 0.01 (original) and 0.001. For all the datasets except for "wisconsin", the model performed best at 0.01 (as in the original code, but without decay). In "wisconsin", the best learning rate was 0.1.

Another hyperparameter has been modified: the number of graph convolutional layers has been set to 64, instead of varying

The rest of the hyperparameters —number of epochs, patience before early stopping, number of hidden layers in the fully-connected layers, dropout, α and λ, are the same as in the original code, in order to recreate at best the original model. The only

4.3 Experimental setup

- GATv2

GATv2 was trained on a free google Colab account.As described in section 3, GATv2 was not trained in an optimal way—all its operations were serialized, hence using a GPU did not help and at times it did deter the performance. So this model was strictly trained on a CPU(Intel(R) Xeon(R) CPU @ 2.20GHz). It used 1.44GB/12.68GB RAM and 38.36GB/107.72GB disk space. Link to the notebook:

- FAGCN

FAGCN was also trained on a free google Colab account. This task was not computationally expensive due to its small size so it was entriely trained on a CPU(Intel(R) Xeon(R) CPU @ 2.20GHz). It used 1.44GB/12.68GB RAM and 38.36GB/107.72GB disk space. Link to the notebook:

- GCNII

The experiments were run using a Google Colaboratory notebook and with the GPU enabled. Link to notebook with instructions: `https://colab.research.google.com/drive/1t-IZEaBunij3-LAPCjILEV2rYdrt0mMG?usp=sharing`.

4.4 Computational requirements

- GATv2

Table 1. .

Bipartite graph size(kxk)	4x4	5x5
GAT-1head	$18'$	$5h$
GATv2-1head	30'	$3h\,59'$
GAT-8heads	$20'$	$1h\,30'$
GATv2-8heads	...	$2h\,30'$

- FAGCN

Training FAGCN on a synthetic dataset was very fast. On an average each training phase only took 18 seconds.

- GCNII

Computational power: an NVIDIA Tesla K80" GPU (the one from Google Colaboratory). The running times for each dataset are shown in Table 1.

Table 2. Running time of GCNII-Spektral per dataset, averaged 3 times.

Datasets	Running time (s)
Pubm	707.15
Corn	17.90
Texa	15.42
Wisc	9.63

5 Results

- GATv2

Table 3. Validation accuracy GATv2

Bipartite graph size(kxk)	4x4	5x5
GATv2-1head	100.0	96.7
GATv2-8heads	93.5	100.0

- FAGCN

Table 4. Validation accuracy FAGCN

Filters $\setminus q$	0.01	0.05	0.07	0.10
LowFreq	98.0	67.0	60.0	66.0
HighFreq	79.0	74.0	91.0	94.0
FAGCN	95.0	85.0	84.0	81.0

- GCNII

In Table 5, one can observe the comparison between the test accuracy of the implementation from the authors of the paper (using PyG) and the one from our implementation (using Spektral).

Table 5. Comparison of the test accuracy between the expected accuracy (the one from the implementation in PyG) and the one from our implementation (Spektral).

Datasets	Acc (PyG)	Acc (Spektral)
Pubm	80.3	60.8
Corn	76.5	70.3
Texa	77.8	57.7
Wisc	81.6	62.8

The only dataset which results are comparable to the original code is Cornell (difference of 6%). The rest present a significant gap between the expected and the obtained accuracies. We attribute this to the fact that we couldn't apply the weight decay (regularisation) for the dense layers.

6 Discussion and conclusion

We conclude that the some of the implementations have gotten results as good as expected or very similar. For example, GATv2 and FAGCN have successfully replicated the models from the papers to Tensorflow-Spektral.

However, the implementation of GCNII has run into challenges that have significantly reduced their performance. As written above, we attribute the discrepancies between the expected results and the ones obtained to the inability to implement the Weight Decay regularisation.

References

[1] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=F72ximsx7C1`.

[2] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *AAAI*. AAAI Press, 2021.

[3] Zhewei Wei Ming Chen, Bolin Ding Zengfeng Huang, and Yaliang Li. Simple and deep graph convolutional networks. 2020.

[4] Ming Chen (chennnM). Github — GCNII: Simple and Deep Graph Convolutional Networks. `https://github.com/chennnM/GCNII/tree/ca91f5686c4cd09cc1c6f98431a5d5b7e36acc92`, 2020. [Online; accessed May-2022].

[5] Namata G. Bilgic M. Getoor L. Galligher B. Eliassi-Rad T. Sen, P. Collective classification in network data. ai magazine, 29(3), 93. 2008.

[6] (User) graphdml-uiuc jlu. Github — Geom-GCN: Geometric Graph Convolutional Networks. `https://github.com/graphdml-uiuc-jlu/geom-gcn`, 2021.