GNN Fundamental - 3

In previous part - we have seen how can we transform the output of GNN embedding to various tasks like node classification, edge classification. Here we will see Graph level tasks.

Graph level task head:

$$\hat{y}_G = \text{Head}_{\text{graph}}\left(h_v^L \in \mathbb{R}^d, \ \forall v \in G\right)$$

We need to design Head graph which will aggregate each node embedding. The aggregated node embedding can be used for downstream task.

Suppose graph has K-class problem where we need to do K-way prediction.

$$\hat{y}_K = \text{Transform}_{\text{Graph}}\left(\hat{y}_G \in \mathbb{R}^d\right)$$
$$\hat{y}_K \in \mathbb{R}^K$$

There will be many ways in which we can Aggregate h_v^L:

① Global pooling:

(A) Global mean pooling: $\text{Mean}\left(h_v^L \in \mathbb{R}^d, \forall v \in G\right)$
coordinate wise take mean.

(B) Global max pooling: $\text{Max}\left(h_v^L \in \mathbb{R}^d, \forall v \in G\right)$
coordinate wise take max value.

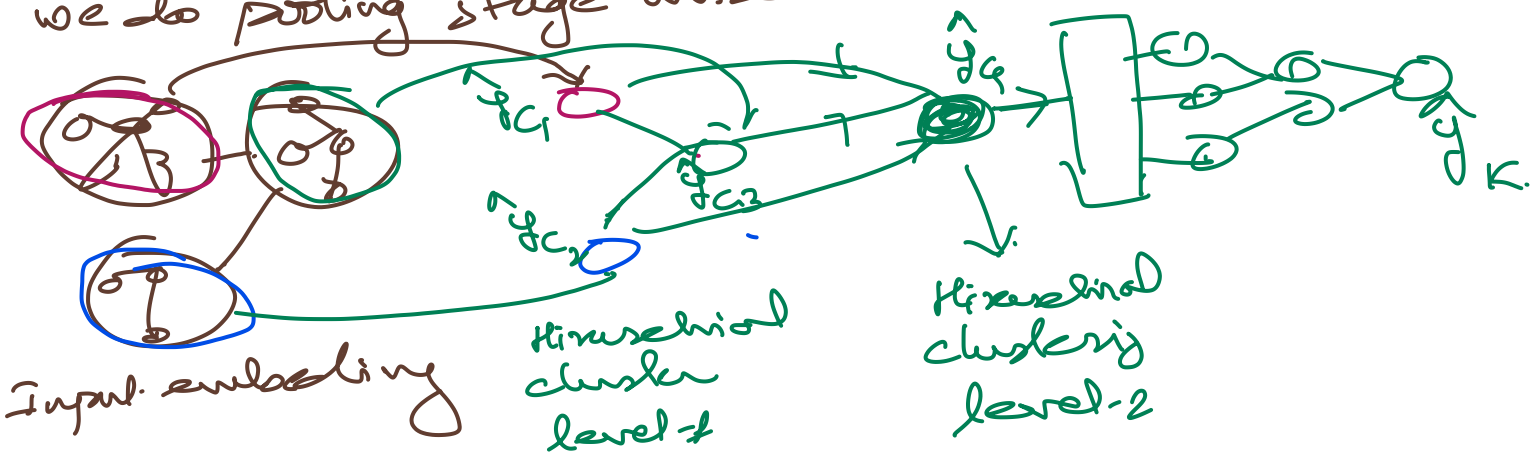
(C) Global sum pooling: $\text{Sum}\left(h_v^L \in \mathbb{R}^d, \forall v \in G\right)$

These option works well with small graph but not for large graph as often we do Global pooling then many details of graph level task. : 1-d embedding.

eg: $G_1 \Rightarrow \hat{y}_{G_1} = \{-1, -2, 0, 1, 2\}$, $G_2 \Rightarrow \hat{y}_{G_2} = \{-10, -20, 0, 10, 20\}$

If we do $\text{Sum}(\hat{y}_{G_1}) = 0 + \text{Sum}(\hat{y}_{G_2}) = 0$ but both graph graph node embedding is different - so graph

will not be same.
For large graph Global pooling is not effective so we do hierarchical pooling. For this we cluster the similar node embedding and for each cluster we do pooling stage wise.



Two GNN used
GNN-A: Compute Node embedding
GNN-B: cluster similar node embedding. This is also hierarchical aggregation of node embedding.
Finally single node embedding remains at top of hierarchi and used for downstream task.

Now we will talk about prediction and labels.
When we take ground truth from external source then that is supervise learning eg taking class label value from external source.
When supervision is missing graph like graph signal for clustering that is called unsupervise or self-supervise eg predicting node clustering coefficient or check if two graph is isomorphic. For this we do not take any external reference for label, so this is unsupervised learning.

Supervised label: This comes from specific use case like in citation network node label y_v will be which subject the node belongs to.

In case of transaction network edge label could be fraud or not. In case of molecular graph TO molecule is toxic or toxicity of molecule.

If we can transform the actual task to node, link or graph level prediction task then we can reuse lot of research, model in this area.

similarly for unsupervised task like predict pagerank, clustering coefficient, predict missing link or what kind of molecule is this? for all this we do not need ground truth but we use graph signal itself to predict these.

Till now we saw prediction and labels. Now we will evaluate discrepancy between prediction and label. For this we define loss function and optimization of loss function will do backpropagation to optimize all learnable parameters. Evaluation Matrix will be used to check performance of model.

Setting up for GNN training:

Let say we have N-data point. This can be for node, edge or graph. Let say:

Prediction	Ground truth	
For node: $\hat{y}_v$	y_v	N-data point can be for any thing. So we say $\hat{y}_i$ & y_i as data point applicable to all.
For edge: $\hat{y}_{uv}$	y_{uv}	
For graph: $\hat{y}_G$	y_G	

class label y_i can be for classification or regression. In classification y_i will be categorical value but in regression it will be continuous value.

eg wether molecule is toxic or not [Binary classification] or what is toxicity of molecule [Regression].

Based on type of problem we define loss function.

classification loss: Most popular loss function is cross entropy to estimate loss when we do classification.

$$CE(\hat{y}_i, y_i) = -\sum_{j=1}^{K} y_i \log(\hat{y}_i)$$

i is i^{th} data point, j = class j of K-class problem.

So $CE(\hat{y}_i, y_i)$ represent cross entropy loss for i^{th} data point

For N data point we need to sum $\sum_{i=1}^{N} CE(\hat{y}_i, y_i)$.

For regression kind of task: Suppose we have K-way regression then we can use mean-square error loss (MSE) for each data point i:

$$MSE(\hat{y}^{(i)}, y^{(i)}) = \sum_{j=1}^{k} \left(\hat{y}_j^{(i)} - y_j^{(i)}\right)^2$$

For all data point

$$Loss = \sum_{i=1}^{N} MSE\left(\hat{y}_j^{(i)} y^{(i)}\right) \Rightarrow \min_{\theta} (Loss) \rightarrow \text{objective function}$$

when loss function optimizes all parameter then we need metrics for evaluation of performance of the model. We use standard evaluation matrix for data like confusion matrix & AOC (Recevier operating curve)

		Actual Class: T	Actual Class: F
Model Predicted class	T	TP	FP
	F	FN	TN

In case of binary classification:

$$\text{Accuracy} = \frac{TP+TN}{\text{Total data set}}$$

$$\text{(P) Precision} = \frac{TP}{TP+FP} = \frac{\text{Actual true out of model predicted true}}{\text{Total Model predicted true class)}}$$

$$\text{(R) Recall} = \frac{\text{Actual true out of model predicted true}}{\text{Total true class.}} = \frac{TP}{TP+FN}$$

$$\text{F1-score(F)} \Rightarrow \frac{1}{F} = \frac{1}{P} + \frac{1}{R}$$

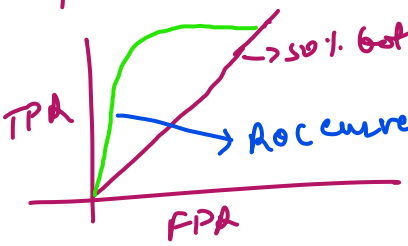
ROC Curve: This is trade-off between true alarm Vs false alarm.

TP is part of true alarm & FP is part of false alarm.

$$\text{TPR(True positive Rate)} = \frac{TP}{TP+FN} = \frac{\text{Actual true out of model predicted true}}{\text{Total actual true class.}}$$

$$\text{FPR(False positive Rate)} = \frac{FP}{FP+TN} = \frac{\text{Model predicted positive which is false}}{\text{Total Negative class.}}$$

So for any model it is important to have high true alarm but at the same time false alarm should be minimum. This is represented as curve (AOC)



Data split for training, evaluation and test: