GNN

KILDER:

https://ai.plainenglish.io/graph-convolutional-networks-gcn-baf337d5cb6b

https://paperswithcode.com/method/gcn

https://towardsdatascience.com/graph-neural-networks-with-pyg-on-node-classification-link-prediction-and-anomaly-detection-14aa38fe1275

Manus:

Welcome to my presentation about GNN in CV.

Firstly what is GNN and why even use it. In conventional CNN the data is usually images with a fixed width and height, and the same amount of pixels, also refered to as **Euclidean** data**.** However because of the structured nature this data is often redundant. Imagine a photo of a road, the entire photo might have a sky, grass, a bird, when in fact we only care about the road. For this we can store this data in a graph, aka all road junction, curves and other features might be a node and we have an edge between all nodes. This will be a much more effective way of storing this data. This is just one example of how to use graphs. Other might be sosial connection, community detection and other cases. Graphs are known as non-eucledian data as they have no structure, there can exists N number of nodes and there can exist edges between some of the nodes, and a node might have an edge to itself as well.

Why not use CNN with graphs?

Graph data is so complex that it's created a lot of challenges for existing machine learning algorithms. The reason is that conventional Machine Learning and Deep Learning tools are specialized in simple data types. Like images with the same structure and size, which we can think of as fixed-size grid graphs. Text and speech are sequences, so we can think of them as line graphs. But there are more complex graphs, without a fixed form, with a variable size of unordered nodes, where nodes can have different amounts of neighbors. It also doesn't help that existing machine learning algorithms have a core assumption that instances are independent of each other. This is false for graph data, because each node is related to others by links of various types.

Where can GNN be usefull:

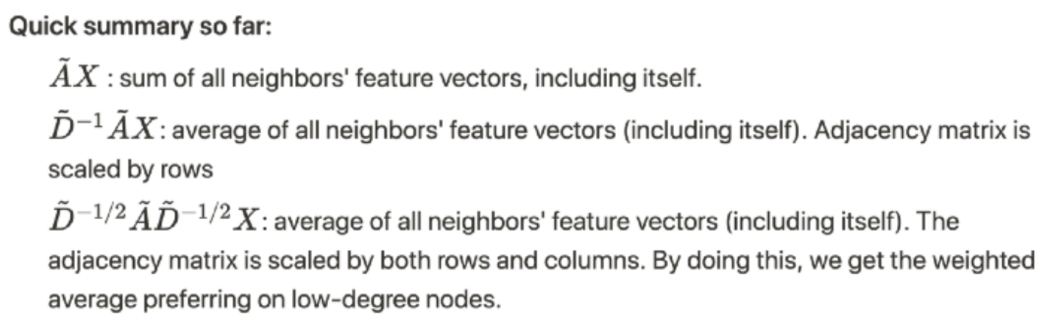
DIFFERENCE BETWEEN SPATION AND TEMPORAL/SPECTRAL

GNN can be used in many contexts, for example node classification:

- **Node Classification**Predicting the classes or labels of nodes. For example, detecting fraudulent entities in the network in cybersecurity can be a node classification problem.
- **Link Prediction**Predicting if there are potential linkages (edges) between nodes. For example, a social networking service suggests possible friend connections based on network data.
- **Graph Classification**Classifying a graph itself into different categories. An example is determining if a chemical compound is toxic or non-toxic by looking at its graph structure.
- **Community Detection**Partitioning nodes into clusters. An example is finding different communities in a social graph.

- **Anomaly Detection**Finding outlier nodes in a graph in an unsupervised manner. This approach can be used if you don't have labels on your target.

How does GNN work:

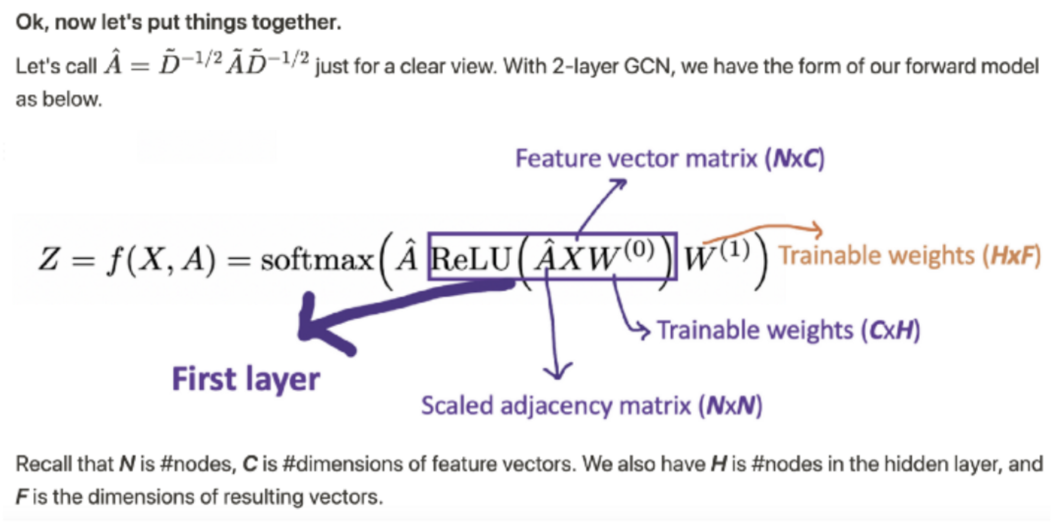


Quick summary so far:

$\tilde{A}X$: sum of all neighbors' feature vectors, including itself.

$\tilde{D}^{-1}\tilde{A}X$: average of all neighbors' feature vectors (including itself). Adjacency matrix is scaled by rows

$\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X$: average of all neighbors' feature vectors (including itself). The adjacency matrix is scaled by both rows and columns. By doing this, we get the weighted average preferring on low-degree nodes.



Ok, now let's put things together.

Let's call $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ just for a clear view. With 2-layer GCN, we have the form of our forward model as below.

$$Z = f(X, A) = \text{softmax}\Big(\hat{A}\ \boxed{\text{ReLU}\big(\hat{A}XW^{(0)}\big)}\ W^{(1)}\Big)$$

Feature vector matrix (**N**x**C**)
Trainable weights (**HxF**)
Trainable weights (**C**x**H**)
First layer
Scaled adjacency matrix (**N**x**N**)

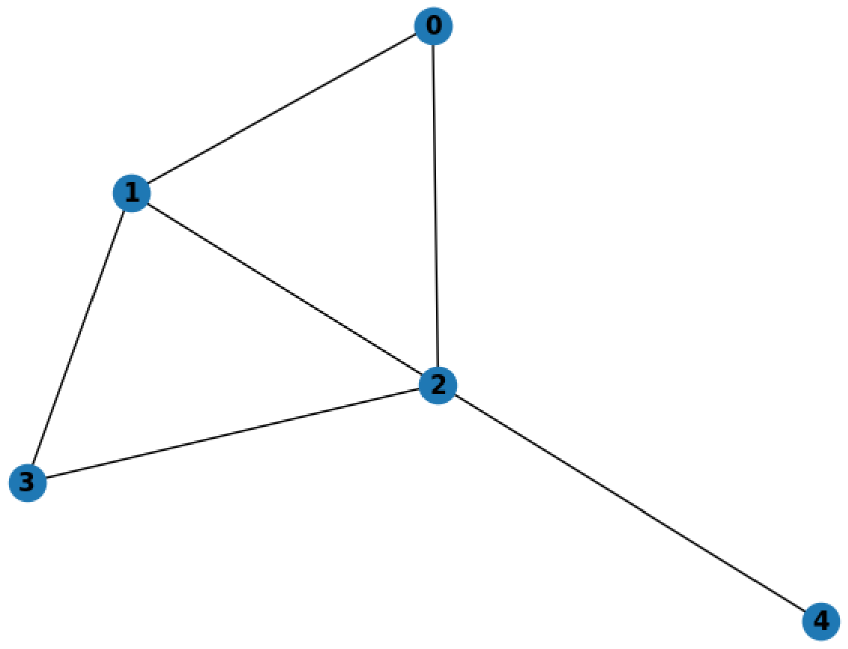
Recall that ***N*** is #nodes, ***C*** is #dimensions of feature vectors. We also have ***H*** is #nodes in the hidden layer, and ***F*** is the dimensions of resulting vectors.

$$H^{[i+1]} = \sigma(W^{[i]}\ H^{[i]}\ A^{*})$$

Equation 3— Forward Pass in Graph Convolutional Networks

WHERE $A^{*} == \hat{A}$

Simple example



Adjacency Matrix (A):

0	1	1	0
1	0	1	1
1	1	0	1
1	1	1	0
0	0	1	0

Node Feature Matrix (X):

0	0	
1	0	
2	1	
3	0	
4	0	

How can we get feature values from heigbors for each node?

Perform Sum of neighboring node features ($A \cdot X$):

3
5
8
3
2

We are still missing the features for the node itself:

We can fix this by adding an identity matrix I to A:

$$\tilde{A} = A + \lambda I_N$$

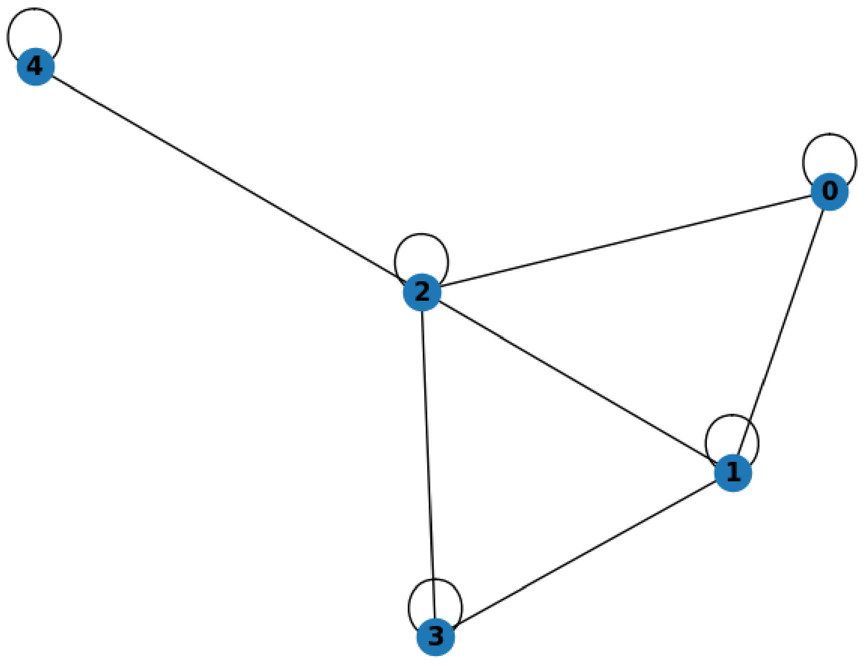
We select $\lambda = 1$ for simplicity

0	1	1	1	0	0	0	0		0	0
1	0	1	0	1	1	0	0		0	0
1	1	0 +	0	1	0	1	1		0	0
1	1	1	0	0	0	0	0		1	0
0	0	1	0	0	0	0	0		0	1

$\hat{A} =$

1	1	1	0	0
1	1	1	1	0
1	1	1	1	1
1	1	1	1	0
0	0	1	0	1

This is equivalent to adding self loops:



$(\hat{A} \cdot X)$:

3

6
10
6
6

Degree matrix ($\hat{D}$)

4	0	0	0	0
0	5	0	0	0
0	0	6	0	0
0	0	0	4	0
0	0	0	0	3

How do we normalize the sum vector? By finding the $\hat{D}\ invese$ and multiplying with X

$\hat{D}^{-1}$:

$\frac{1}{4}$	0	0	0	0
0	$\frac{1}{5}$	0	0	0
0	0	$\frac{1}{6}$	0	0
0	0	0	$\frac{1}{4}$	0
0	0	0	0	$\frac{1}{3}$

GCN Task

GCN Solution