GRAPH NEURAL NETWORKS

- **PRESENTED AS A GRADUATION PROJECT REQUIREMENTS FOR ECE 6143 INTRODUCTION TO MACHINE LEARNING.**
- **BASED ON PAPER : A COMPREHENSIVE SURVEY ON GRAPH NEURAL NETWORKS, ZONGHAN WU ET AL. - 2019 .**
- **IMPLEMENTATION IS BASED ON : SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS, KPIF ET AL. – 2017**
- **PRESENTED BY STUDENTS :**
  - **Hafez ElBoghdadi**
  - **Kadidajatou Mossi**

AGENDA

1. Graphs Overview
2. Graphs Embeddings
3. Graphs in Machine Learning
4. Graph Neural Networks Vs Neural Networks in Machine Learning
5. Machine Learning Tasks in Graph Neural Networks
6. Graph Neural Networks Limitations.
7. GNN drawbacks
8. Types of Graph Neural Networks
9. Main Theory of Convolutional Graph Neural Networks
10. Validating a Claim in Kpif’s ConvGNN
11. Conclusion
12. References

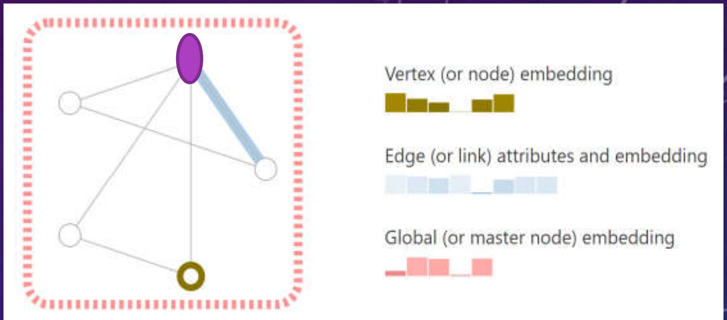
GRAPHS OVERVIEW

1.Graphs : A mathematical structure representing a connection between different or similar objects

1. Networks of roads covering a city.
2. Friends of Friends on Social Media.
3. An electric circuit composed of interconnected elements.

2.All graphs are represented by ;

1. The Whole Graph : May carry some significance, different graphs represent different chemical compounds.
2. Vertices : The nodes or points, representing an object like cities, persons, atoms, These vertices can carry information or none ; For Molecules the vertices can represent a "C" or "H" atom, for a social network the vertices can carry a username Or can carry just an ID number if the network is anonymized.
3. Edges : The connection between nodes, for example NY is directly connected to Jersey City through a road, however it is connected to London through a flight path but no direct roads, thus we can consider NY and Jersey City are connected by an "edge" (road) but isolated from London (or have a high-cost "edge" using airplanes)
    1. The connection between vertices and edges in practice are mostly 0s and 1s → Sparse Matrices
    2. Edges can be directed (Person A is traveling to Person B's Location) Or bidirectional (A is a friend of B -implies- that B is also a friend of A "usually")
    3. Edges can have attributes like cost or degree of strength (Cost of airplane travel "edge" is higher than a train "edge") or without any attributes (i.e. all roads are equal in cost and time).
    4. Edges can have multiple Representations : For a graph G with |V| Nodes.
        1. Adjacency Matrix A : |V| * |V| Matrix such that [A] a_ij = "c" when there is an edge between i → j with weight "c" [Less Efficient in Storage]
        2. Common Coordinate Format : Edges are represented by 3 lists ; A list for source nodes, a list of corresponding destination nodes, and a data list indicating the weight/attribute of each edge [More efficient]
        3. Adjacency List : An ordered list of the edges destinations per node, for example if the ith- element of the list is (j,k,l) ordered pairs, this mean that the i-th node has 3 edges from i→j , i→ k, i→ l
4. Graph Metrics : These metrics determine the connectivity strength between nodes, for example **Degree Centrality** will indicate how many edges are connected to each node.
    1. The More a node is connected to other nodes in the graph, the higher its degree - requiring normalization later to avoid the dominance of this node features in calculations.

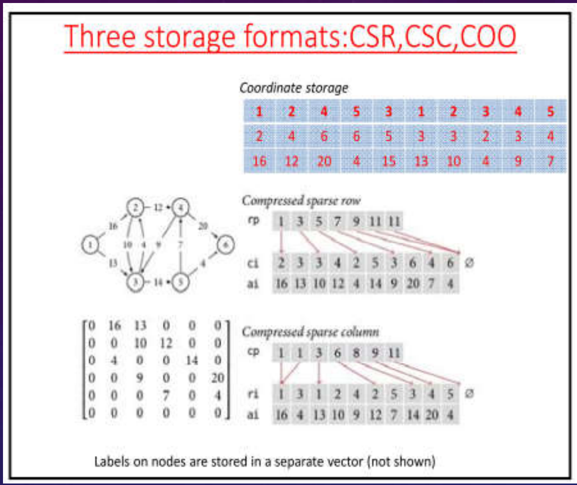

Vertex (or node) embedding
Edge (or link) attributes and embedding
Global (or master node) embedding

Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).

- Each Node can carry a signal (attributes), also called embedding which is a vector representation of the node, the same is applicable to the edges and the whole graph.
- Ex: Nodes are the students, Node attributes are their degrees in different subjects, edges are the roads connecting their hometowns, edge attributes are the roads traffic rate in different times of the day, the whole graph represents the friendships between a group of students in a project, and the graph attributes are the group's performance evaluation as a team not as individuals in different projects.
- A sample graph, where the violet node has higher centrality than other nodes in this graph.

https://distill.pub/2021/gnn-intro/

GRAPHS OVERVIEW



Three storage formats:CSR,CSC,COO

Coordinate storage

1	2	4	5	3	1	2	3	4	5
2	4	6	6	5	3	3	2	3	4
16	12	20	4	15	13	10	4	9	7

Compressed sparse row

```
rp  1  3  5  7  9  11 11
ci  2  3  3  4  2  5  3  6  4  6  ∅
ai  16 13 10 12 4  14 9  20 7  4
```

$$\begin{bmatrix} 0 & 16 & 13 & 0 & 0 & 0 \\ 0 & 0 & 10 & 12 & 0 & 0 \\ 0 & 4 & 0 & 0 & 14 & 0 \\ 0 & 0 & 9 & 0 & 0 & 20 \\ 0 & 0 & 0 & 7 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

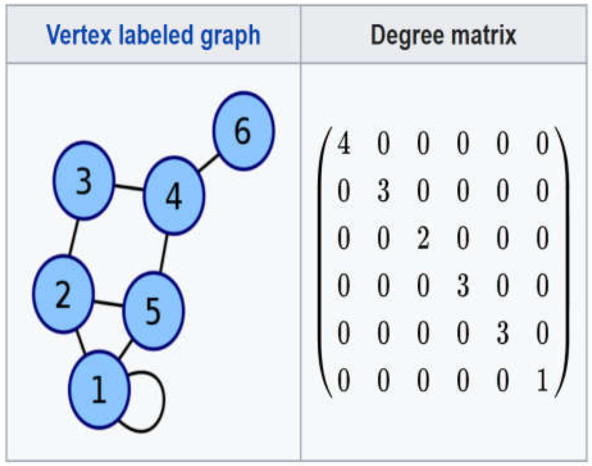
Compressed sparse column

```
cp  1  1  3  6  8  9  11
ri  1  3  1  2  4  2  5  3  4  5  ∅
ai  16 4  13 10 9  12 7  14 20 4
```

Labels on nodes are stored in a separate vector (not shown)

A weighted directed graph with its COO and Adjacency Matrix representation.

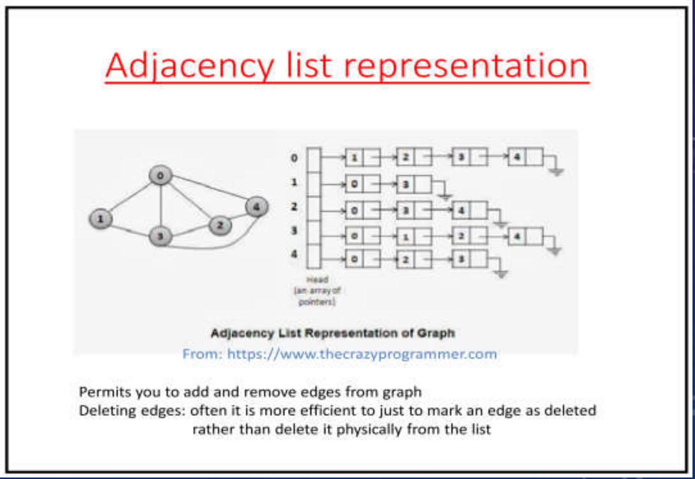
https://www.cs.utexas.edu/~pingali/CS377P/2018sp/lectures/GraphAlgorithms-quad.pdf



$$\begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The Diagonal Degree Matrix of a graph has the value of d+1 , where d is the degree of each node .

https://en.wikipedia.org/wiki/Degree_matrix


Adjacency list representation
Head (an array of pointers)
Adjacency List Representation of Graph
From: https://www.thecrazyprogrammer.com

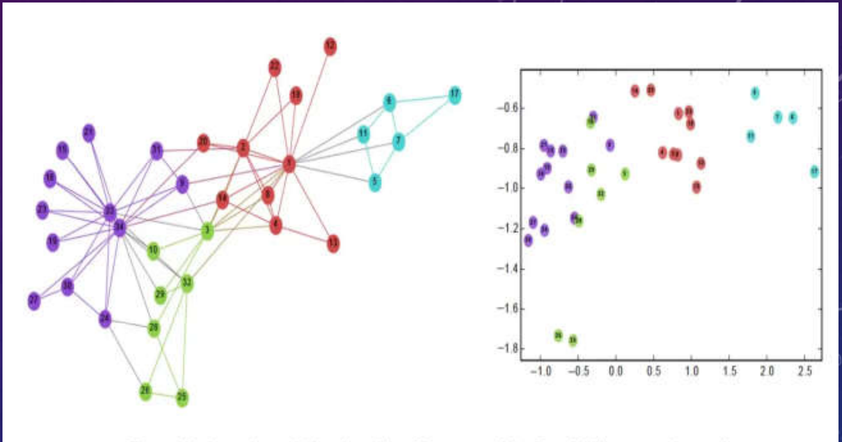
Permits you to add and remove edges from graph
Deleting edges: often it is more efficient to just to mark an edge as deleted rather than delete it physically from the list

The Adjacency List of a unidirectional graph

https://www.cs.utexas.edu/~pingali/CS377P/2018sp/lectures/GraphAlgorithms-quad.pdf

GRAPHS EMBEDDINGS

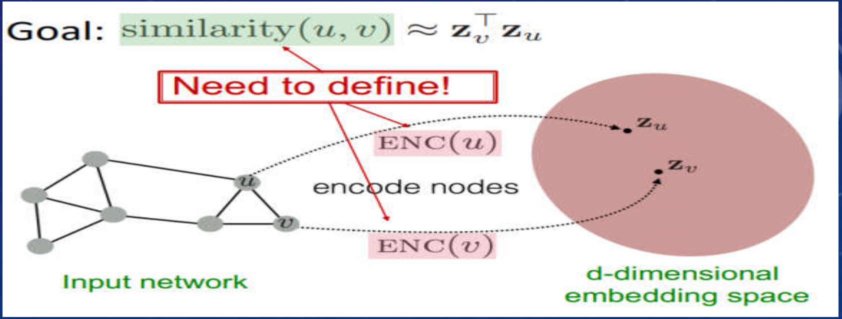
- Graph Embedding is the process of using different algorithms to map the graph elements from their graph form into a lower-dimensional vector.
- The point is to be able to get for example a 2-D mapping f: G→ R2 for either one of ;
  - The whole graph.
  - The edges or the nodes.
  - Or the nodes to a let us say.
- The embedding should reflect the nature of the graph as soon as possible, nodes closer to each other and highly connected should have their location vectors closer to each other.
- The algorithms are categorized into 3 main categories , they differ in their capability to embed new node/graph with or without training, i.e. some algorithms will require training on a group and graphs and can provide an embedding relative to the input graph.
  - Transductive Embedding: If a new graph is introduced then re-training is required
  - Inductive Embedding: If a new graph is introduced then an embedding can be generated without requiring any new training.
- Closeness of embedding is determined by some functions, most common of which is the L2 distance between 2 ndoes embeddings , expected to be small for closer nodes and vice versa.



The original graph and it's embedding (Courtesy of the DeepWalk research team)

https://towardsdatascience.com/overview-of-deep-learning-on-graph-embeddings-4305c10ad4a4
https://medium.com/@st3llasia/graph-embedding-techniques-7d5386c88c5

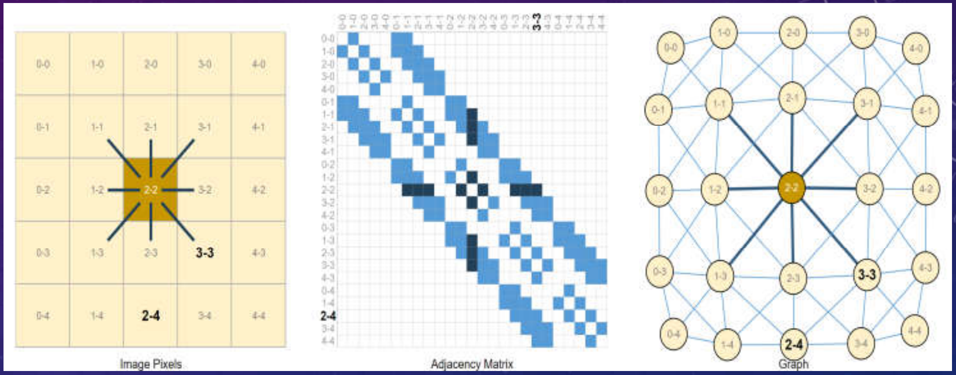
- DeepWalk and Node2Vec are both examples of Transductive "Nodes Embeddings" algorithms that rely on random walk algorithms to estimate the importance and "closeness" of a node to other groups of nodes, the more frequent a node I is present in multiple short-length (3-edges walk for example) random walks starting from node J throughout the graph the more expectation that this node I is close to node J when mapped to the lower dimensional vectors.
- The simple pseudo implementation will be discussed later.



https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications
https://snap.stanford.edu/node2vec/

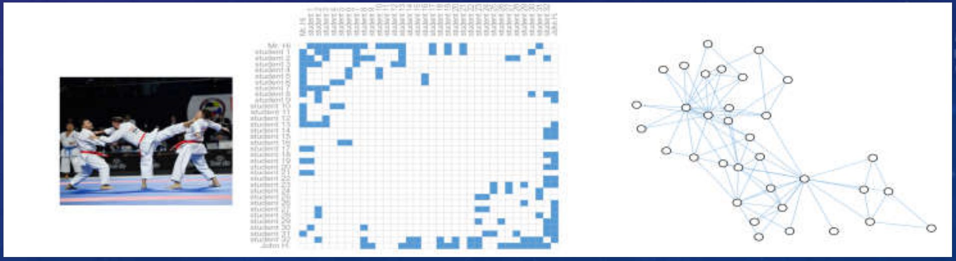
GRAPHS IN ML

- Within the classic ML context, Graphs can be used to represent well-known data forms like images, where each pixel can be seen as a node, in this case the adjacency matrix is symmetric since the image is mapped from 2D and all pixels are by definition joined.
- For images, we already know about Conv Neural Networks, so Why GNN ?
- Some Other datasets or problems in nature are better represented as graphs, in our project we used one of them, a social network dataset known as Zachary's Karate club



Pixel (2,2) is connected to 8 adjacent pixels, each represented by a node, and their adjacency is represented by the edges connecting them

https://distill.pub/2021/gnn-intro/



Zachary's Karate Club dataset represented by its adjacency matrix and nodes

https://distill.pub/2021/gnn-intro/
http://konect.cc/networks/ucidata-zachary/

GNNS VS NNS IN ML

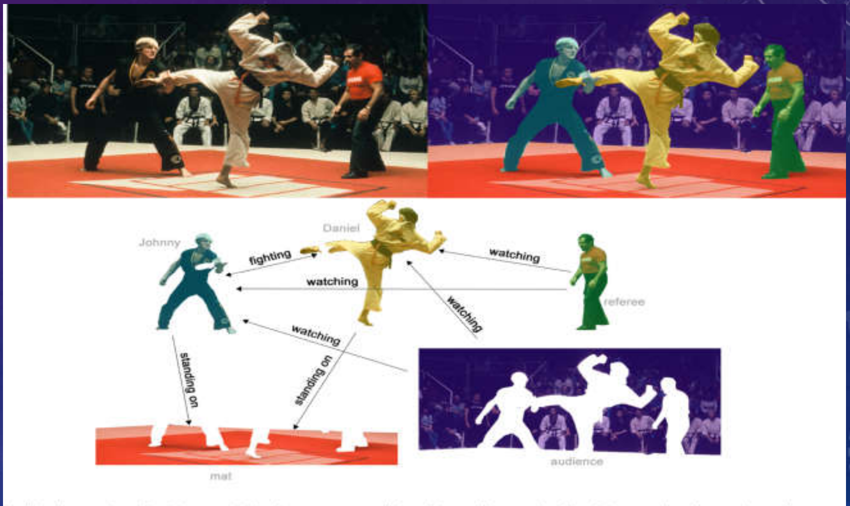
- GNNs are more suited to Graph-based structures than conventional Neural Networks.
  - Images can be considered only as a special case of graphs where every node is labeled, ordered and connected via a symmetric adjacency matrix with each of its surrounding nodes in a grid form.
  - Graphs in general do not need to be ordered or symmetric, just as the case in Social Networks Datasets (ex; Zachary's Karate Club , Citations between different publications).
  - CNNs will require to traverse on the entire graph in order to localize each area, which is computationally ineffective, while in Graphs this is can be done directly using the Graph Nodes attributes along with the graph adjacency matrix.
  - Graphs do not have a unique adjacency matrix representation, the same graph can have multiple adjacency matrices depending on the nodes order.

https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications
https://analyticsindiamag.com/why-graph-neural-networks-are-gaining-popularity-in-2021/

- Among the common GNN applications ;
  - Google Maps ETA Calculation.
  - UberEats Recommendation Systems.
  - NLP : Text represented as graphs.

GNNS VS NNS IN ML

- One of the famous examples of the difference between Conv Neural Networks and Graph Neural Networks is the “Edge Inference”
- ConvNN can perform image processing tasks like humans detections, but cannot identify what activities are these humans actually doing in the photos.
- If you can represent the humans as graphs, you can infer their activities/feelings/relations with one another when represented as graphs with paying special attention to the eyes and the body gesture
- A raised leg will indicate a fighting stance, a relaxed stature will indicate a passive or a “watcher” role like the referee.



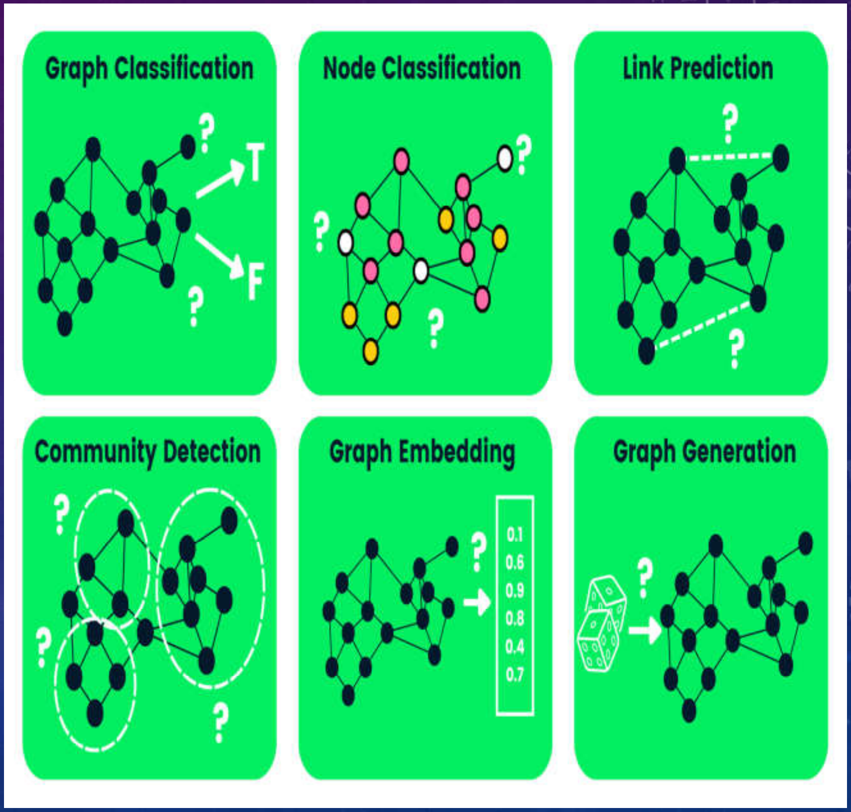
In (b), above, the original image (a) has been segmented into five entities: each of the fighters, the referee, the audience and the mat. (C) shows the relationships between these entities.

https://distill.pub/2021/gnn-intro/
[A shot from Karate Kid 1985, the final duel scene)

ML TASKS IN GNNS

- GNNs are used for many problems when represented as graphs ;

1. Graph Class : Is this compound effective against cancer.
2. Node Class : My friends are members of which club based on my and their own clubs memberships.
3. Link Prediction : Is there a connection between 2 people who are friends in reality yet they are not facebook friends according to my dataset.
4. Community Detection : Similar to unsupervised clustering but for Graphs, like determining an instagram influencer’s impact area.
5. Graph Embedding : What is the approximate numerical closeness between each node.
6. Graph Generation : If Compounds A and B have 70% and 80% regenerative capabilities but with negative 50% and 90% withdrawl rates, can we synthesize a compound having the best traits of the A and B ?



From <https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial>

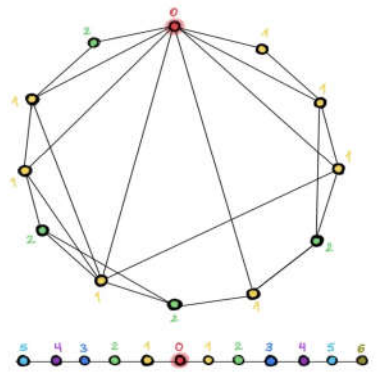
GNNS LIMITATIONS

- Like any ML techniques ; there are multiple drawbacks of GNNs ;

1. Shallow By Nature : The performance of the GNNs with more than 3-layers either lowers or does not provide any performance enhancement for the accuracy.

This is mainly since each node can know about its surroundings within few layers covering the next-hops.

Example on the left : the red node will learn about the yellow nodes in the first layer, green in second, blue in 3rd and so on, if we kept increasing the layers then more and more features from the neighbors will be “squashed” over into the single red node, while in actuality in this graph you would need only 2 hops to reach any other node so 2-layers are enough for every node to get info about the whole graph.



In small-world graphs (top), only a few hops are needed to reach any node from another one. As a result, the number of neighbours (and accordingly, the receptive field of the graph convolutional filters) grows exponentially fast. In this example, only two hops are needed in order to each any node from the red one (different colours indicate the layer in which the respective node will be reached, starting with the red one). On grids (bottom), on the other hand, the growth of the receptive field is polynomial and thus many more layers are needed to reach the same receptive field size.

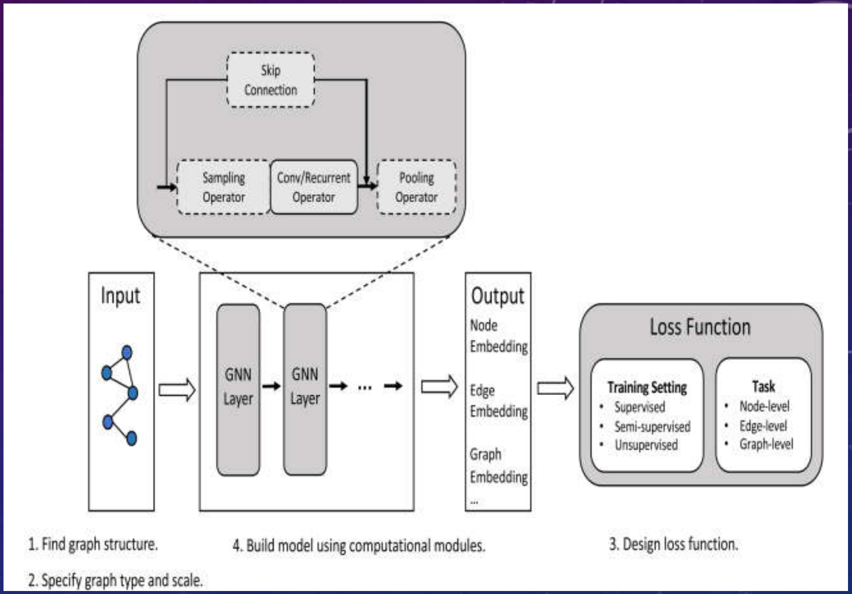
Do we need deep graph neural networks?
https://towardsdatascience.com/do-we-need-deep-graph-neural-networks-be62d3ec5c59
Limitations of Graph Neural Networks
-https://wandb.ai/syllogismos/machine-learning-with-graphs/reports/18-Limitations-of-Graph-Neural-Networks--VmlldzozODUxMzQ

2. Graph pooling is used occasionally using average or sum functions since they are Permutation Invariant (do not change when the input is shuffled), this could cause information loss since each node receptive field is limited to its neighbors.

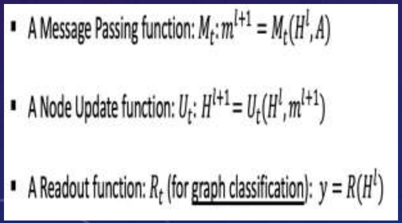
The Essential Guide to GNN (Graph Neural Networks) - HTTPS://CNVRG.IO/GRAPH-NEURAL-NETWORKS/
Oxford Protein Informatics Group
- https://www.blopig.com/blog/2021/10/issues-with-graph-neural-networks-the-cracks-are-where-the-light-shines-through/

TYPES OF GRAPH NEURAL NETWORKS

- The survey paper categorized the GNNs according to multiple criteria.
- Each GNN starts with an initial State of a Matrix X of all node vectors stacked row per node + A matrix represents the graph structure usually the adjacency matrix containing the edges weights if there are any.
- There are main functions/building blocks.
- for a Node i, its initial state (attributes) vector x_i, and its Neighbor nodes N(i), their attributes (information) vectors x_j where j belongs to {N(i)} :
    1. Propagation : Updates Node i with the information from N(i).
       Also can be further divided Message Passing and Node Update functions.
    2. Sampling : Optional – To sample from N(i) which set of nodes N`(i) that will be used in the propagation.
    3. Pooling : Optional – Similar to Conv NN, examples are max/average pooling to aggregate the neighbor nodes attributes
    4. An activation function applied at the output.
- The nodes features vectors are known as X state matrix , or H(0) or aka Layer-0 Latent Matrix , where each row represents the node feature input to the GNN.
- The output of each layer of GNN transforms X == H(0) from a |V| x P feats Matrix into a latent matrix H(k) where each row represents the latent or transformed nodes features.

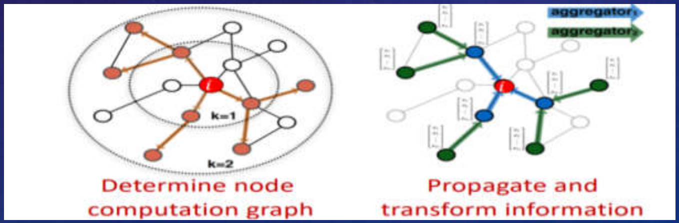


Graph Neural Networks : A Review of Methods and Applications , Jie Zhou et al. - 2020
https://arxiv.org/ftp/arxiv/papers/1812/1812.08434.pdf

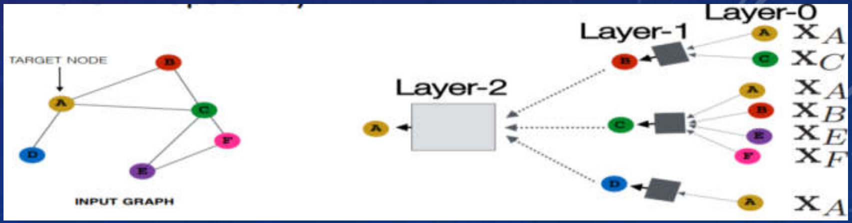


- A Message Passing function: $M_t: m^{l+1} = M_t(H^l, A)$
- A Node Update function: $U_t: H^{l+1} = U_t(H^l, m^{l+1})$
- A Readout function: R_t (for graph classification): $y = R(H^l)$

Any Graph Neural Network can be expressed as a Message Passing Neural Network (J. Gilmer et al. , 2017) with a ***message-passing*** function, a ***node update*** function and a ***readout*** function.
https://towardsdatascience.com/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cfdc


k=1
k=2
aggregator
aggregator
Determine node computation graph
Propagate and transform information

In Layer K=1 , each node I will receive info from its adjacent nodes (1st-hop), at layer K=2, the same node will receive info from the same adjacent nodes however these adjacent nodes already got info from their neighbors, so it is like the node I received info from the 2nd-hop nodes, and so on for each layer.
https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications
Graph Neural Network and Some of GNN Applications: Everything You Need to Know



A Figure indicating a 2--layer Graph NN, Node A will receive information from 2nd-hop neighbors since each layer will provide info from a 1st-hop layer, the

https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications
Graph Neural Network and Some of GNN Applications: Everything You Need to Know
Photo is taken from Stanford course : https://web.stanford.edu/class/cs224w/slides/08-GNN-application.pdf

TYPES OF GRAPH NEURAL NETWORKS

- There are 2 main taxonomies ;
  - Zonghan et al. 2019
  - Jie Zhou et al. 2020
- The main outcome is to get a node embedding (our focus was on nodes embeddings not in edge/graph embeddings) that closely represents the graph structure relationship with the output true label Y vector per node.
- The main target is aggregate and accumulate nodes information between each nodes until an equilibrium state is achieved with a minimal loss function

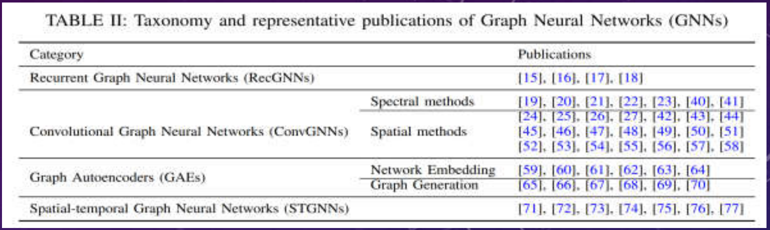


TABLE II: Taxonomy and representative publications of Graph Neural Networks (GNNs)

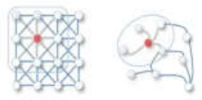
Category		Publications
Recurrent Graph Neural Networks (RecGNNs)		[15], [16], [17], [18]
Convolutional Graph Neural Networks (ConvGNNs)	Spectral methods	[19], [20], [21], [22], [23], [40], [41]
	Spatial methods	[24], [25], [26], [27], [42], [43], [44] [45], [46], [47], [48], [49], [50], [51] [52], [53], [54], [55], [56], [57], [58]
Graph Autoencoders (GAEs)	Network Embedding	[59], [60], [61], [62], [63], [64]
	Graph Generation	[65], [66], [67], [68], [69], [70]
Spatial-temporal Graph Neural Networks (STGNNs)		[71], [72], [73], [74], [75], [76], [77]

A Comprehensive Survey on Graph Neural Networks, Zonghan et al. - 2019
https://arxiv.org/pdf/1901.00596.pdf

A. Spectral Methods:

- Each node latent feature vector is a function of the neighbor's latent feature vectors normalized by the graph degree.
- Mathematically similar to an approximated Convolution.
- Different Weights in each layer.

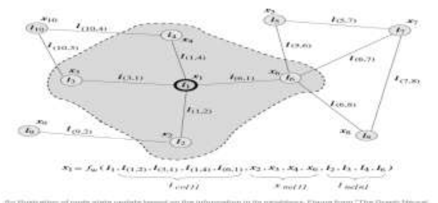
B. Spatial Methods



Left: Convolution on a regular graph such as an image. Right: Convolution on the arbitrary graph structure. Figure from "A Comprehensive Survey on Graph Neural Networks"

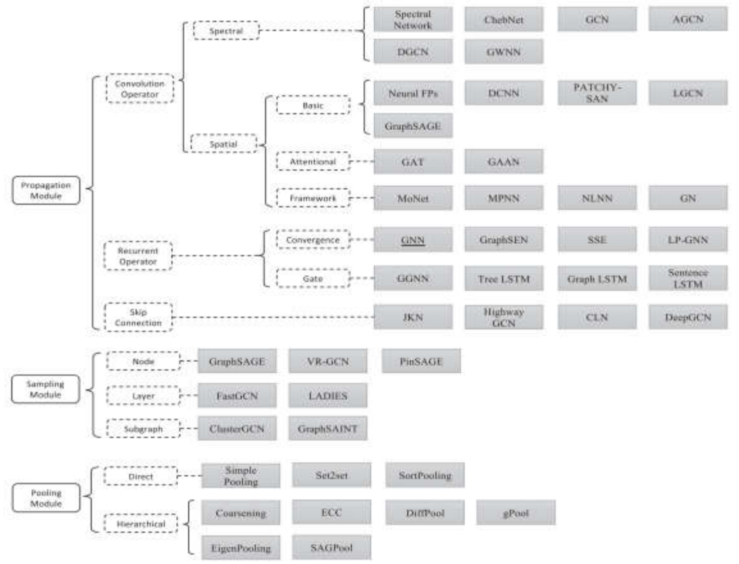
- Similar to ConvNN but applied on a graph as an extension, the convolution window is traversed along the graph nodes.
- Different Weights in Each Layer as well.

C. Recurrent GNNs



- The latent node features for a node in any stage is a function f_w of All of the following ; Its own current latent features, the edges features (which is usually not present in ConvGNN , the neighbors latent features, and the neighbors input features (original state) .
- The o/p is passed by an nonlinear activation function of the latent and original states of each node to yield a node embedding representation.
- Weights are shared in all stages.

RecGNN Example – from An Introduction to Graph Neural Network(GNN) For Analyzing Structured Data
https://towardsdatascience.com/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cfdc



Graph Neural Networks : A Review of Methods and Applications , Jie Zhou et al. - 2020
https://arxiv.org/ftp/arxiv/papers/1812/1812.08434.pdf

We choose to implement the ConvGNN using Spectral Methods, since it was the basis and the main building block for Spectral GNNs .

MAIN THEORY OF CONV GNN

- The implementation details are listed in **Semi-Supervised Classification with Graph Convolutional Networks -** Kpif et al. - 2016

Algorithm :

A) Given : K-Layers , k=1,2,.. K

- Graph G~(V Nodes, E Edges)
- Edges represented by an Adjacency Matrix A (|V|, |V|)
- Each Node i has an initial attribute vector x_i (1, D) , Stacked row by row giving the Features Matrix X (|V|, D)
- h_k : Hidden Feature Map dimension at layer k. [In the paper it is H but we will use h to avoid confusing the dimensions with the hidden signal H
- f : Output Layer Dimension
- For Layer k, the input is H[k-1] , and the output is H[k], the output of the last layer is H[K] = Z

B) Initialization :

1. Assume X = H[k=0] .
2. Initialize Weights Matrices for each Layer using random numbers or better using orthogonal Weight matrices defined on "Understanding the difficulty of training deep feedforward neural networks" – Xavier Golot et al. – 2010 , the Weight matrices will be defined as ;
    - W[1] is a (D, h_1) Matrix , W[2] is a (h_1, h_2) Matrix, W[3] is a (h_2, h_3) and so on until the last layer W[K] is a (h_(k-1) , F) Matrix .
    - Take a note that the weight matrices dimensions do not necessarily need to be equal in size
3. From the Adjacency Matrix; Generate A^ = A + I_|V| , where I is the identity matrix with dim (|V|, |V|) , A^ represents the adjacency matrix and a self-loop connecting each node to itself through a edge loop.

   Why ? This is to take the node's own features into consideration when the propagation equation is calculated, otherwise it will include only the features of the neighboring nodes only.

4. Calculate the Diagonal Degree Matrix D from A^ , where D = Diag (Degree of node i)
Why ? This will be used to normalize the output of the propagation equation, to avoid vanishing or exploding weights when a node has a strong centrality measure (has too many edges) so its features can easily overwhelm the rest of the nodes).

5. Calculate A~ = D^(-1/2) A^ D^(-1/2) , this is the normalized version of the adjacency matrix, used as a trick within the paper.
The left and right multiplication is done to accommodate for the inbound and outbound edges in case of directed graphs, and it has the same effect of multiplying the node's attributes by 1/sqrt(No. of node i neighbors * number of node j's neighbors) for each (i,j) element in the matrix.

6. The output of the first GCN Layer is calculate as ;

   H[1] = A~ * H[0] * W[0] = A~ X * W[0] = D^(-1/2) A^ D^(-1/2) * X * W[0]

7. The output can be applied to non-linear activation function like ; Tanh, LeakyRelu , Relu did not give good performance results due to the vanishing gradient mostly.

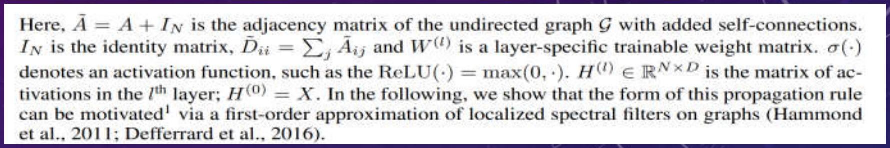
   H`[1] = Tanh (H[1])

8. And repeat for each layer , H[2] = A~ * H`[1] * W[1]
9. In the final output stage, apply a classifier function and a loss function depending in the problem at hand (Softmax and Cross Entropy) :

   Z = Softmax (H`[K])

MAIN THEORY OF CONV GNN

Given

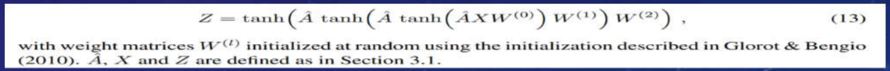


Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, such as the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$. $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$. In the following, we show that the form of this propagation rule can be motivated[1] via a first-order approximation of localized spectral filters on graphs (Hammond et al., 2011; Defferrard et al., 2016).

The Propagation Equation

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right)$$

Ex: 3-Layers GCN with Tanh Activation



$$Z = \tanh\left(\hat{A} \tanh\left(\hat{A} \tanh\left(\hat{A} X W^{(0)}\right) W^{(1)}\right) W^{(2)}\right) , \tag{13}$$

with weight matrices $W^{(l)}$ initialized at random using the initialization described in Glorot & Bengio (2010). $\hat{A}$, X and Z are defined as in Section 3.1.

Ex: 2-Layers GCN with Softmax Output and Tanh Activation

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A} \ \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) W^{(1)}\right)$$

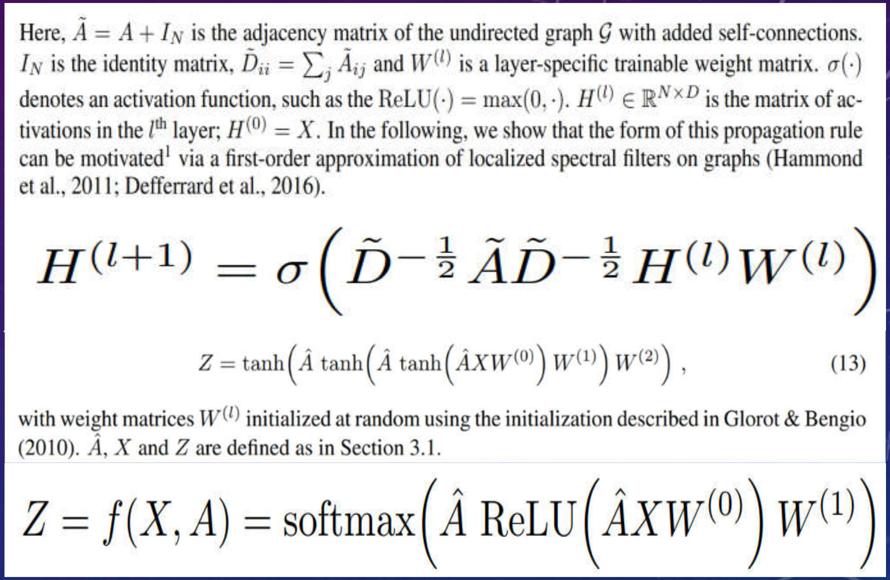
CLAIM TO BE PROVEN WITHIN THE PAPER

Given

The Propagation Equation

Ex: 3-Layers GCN with Tanh Activation

Ex: 2-Layers GCN with Softmax Output and Tanh Activation



Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ 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 $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$. $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$. In the following, we show that the form of this propagation rule can be motivated[1] via a first-order approximation of localized spectral filters on graphs (Hammond et al., 2011; Defferrard et al., 2016).

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right)$$

$$Z = \tanh\left(\hat{A} \tanh\left(\hat{A} \tanh\left(\hat{A} X W^{(0)}\right) W^{(1)}\right) W^{(2)}\right) , \quad (13)$$

with weight matrices $W^{(l)}$ initialized at random using the initialization described in Glorot & Bengio (2010). $\hat{A}$, X and Z are defined as in Section 3.1.

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) W^{(1)}\right)$$

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS – Kpif et al. 2017
https://arxiv.org/pdf/1609.02907.pdf

VALIDATING A CLAIM IN KPIF’S CONVGNN

Claim : Un-trained ConvGNN can provide comparable Node Embedding for the nodes in Zachary’s 1977 Graph dataset against using the computationally intensive DeepWalk Node Embedding Technique.
Listed in Page 12 in the paper.

Dataset : Zachary’s Karate Club , a graph that nodes → students, edges → interactions after class , each student is a member of one of 4 rival Karate clubs (In the original dataset they were actually just 2 but this paper used a modified version of the dataset with the 4 clubs)
The Dataset is used within a semi-supervised context, we know the true membership of each student, but we will train and calculate the loss function using only 4 (or 2 if using the original dataset) nodes only, 1 for each class member.

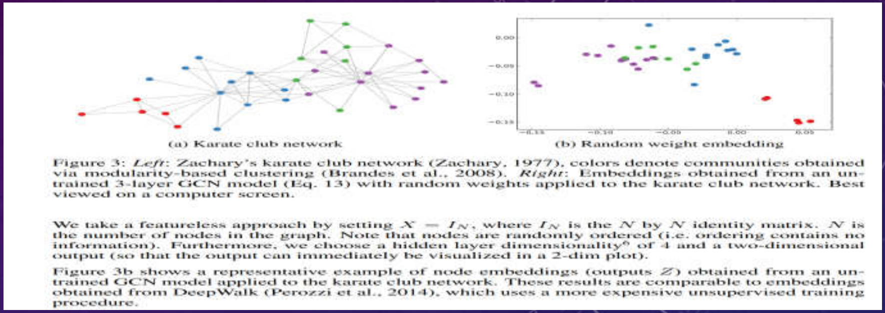

(a) Karate club network
(b) Random weight embedding

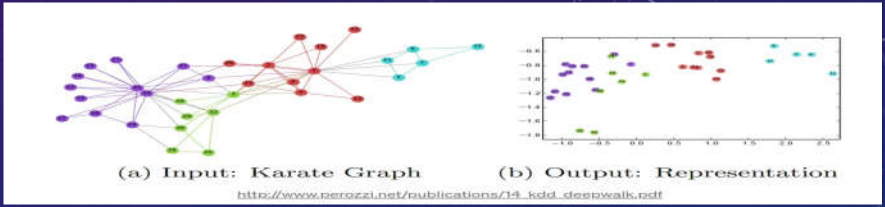
Figure 3: *Left*: Zachary’s karate club network (Zachary, 1977), colors denote communities obtained via modularity-based clustering (Brandes et al., 2008). *Right*: Embeddings obtained from an un-trained 3-layer GCN model (Eq. 13) with random weights applied to the karate club network. Best viewed on a computer screen.

We take a featureless approach by setting $X = I_N$, where I_N is the N by N identity matrix. N is the number of nodes in the graph. Note that nodes are randomly ordered (i.e. ordering contains no information). Furthermore, we choose a hidden layer dimensionality[6] of 4 and a two-dimensional output (so that the output can immediately be visualized in a 2-dim plot).

Figure 3b shows a representative example of node embeddings (outputs Z) obtained from an un-trained GCN model applied to the karate club network. These results are comparable to embeddings obtained from DeepWalk (Perozzi et al., 2014), which uses a more expensive unsupervised training procedure.

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS – Kpif et al. 2017 - https://arxiv.org/pdf/1609.02907.pdf

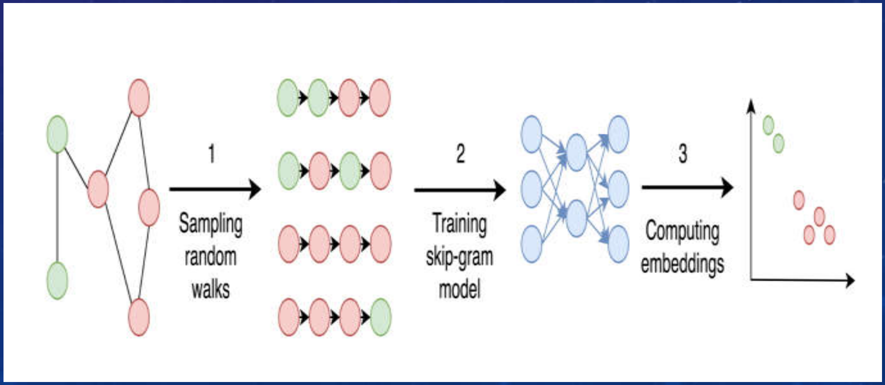
Paper Implementation :
a) The authors used an untrained 3-layers GCN with Tanh and a final SoftMax classifier with Cross-Entropy Loss, Hidden Feature Map dim H = 4 and Output Map dim = 2
b) The authors plotted the 2-dim Z –signal SoftMax o/p in R2 Vector space for an untrained , their plot semi-segregated the rival clubs members in different clusters and grouped the same club members closer to one another, without needing too many computations compared to the DeepWalk algorithm



A Gentle Introduction to Graph Neural Networks (Basics, DeepWalk, and GraphSage) https://towardsdatascience.com/a-gentle-introduction-to-graph-neural-network-basics-deepwalk-and-graphsage-db5d540d50b3
DeepWalk: Online Learning of Social Representations – Perozzi et al. - http://www.perozzi.net/publications/14_kdd_deepwalk.pdf

DeepWalk : An algorithm for Node embeddings; several parameters are defined ;
a) “t” Length of the random walk : The number of edges to be covered within a path.
b) “d” Embedding size : the dimensions of the reduced-dim vectors representation of the nodes, =2 or 3 for interpretability in R2 or R3 Vector Spaces.
c) “y” Walks per vertex : How many random walks to be generated per each vertex.
d) “w” Window Size : The window width containing the w/2 number of vertices before and after the originating vertex walk.
e) “lr” Learning Rate : To be used in the GNN to train the random walk algorithm neural network.

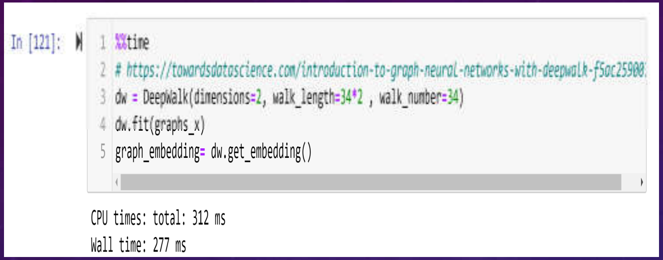
A Walk is a path traversed across the graph from one adjacent node to another starting from a specified originating vertex , yielding a set of random paths or walks.
These walks –without going into much details for time-constraints- are used to train an NLP-inspired NN called Skip-gram model to yield a 2-d or 3-d representations for each node after a series of computations.



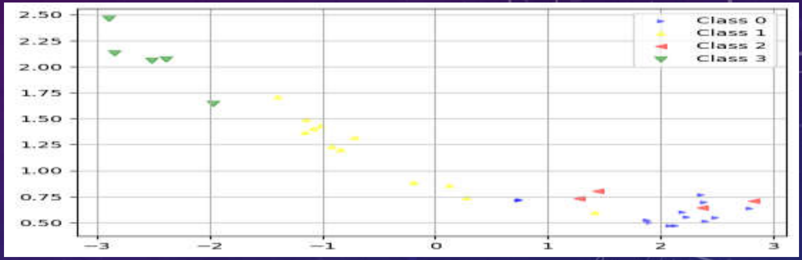
UNDERSTANDING DEEPWALK -DeepWalk https://dsgiitr.com/blogs/deepwalk/
Code Used From Introduction to Graph Neural Networks with DeepWalk https://towardsdatascience.com/introduction-to-graph-neural-networks-with-deepwalk-f5ac25900772

RESULTS OF THE CLAIM VALIDATION – SAMPLE RUNS

DeepWalk – Non-supervised (No ouput labels are provided)



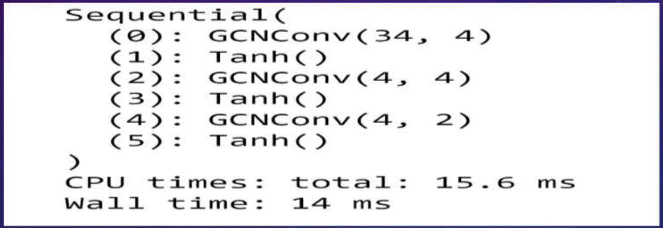
```
In [121]:  %%time
           # https://towardsdatascience.com/introduction-to-graph-neural-networks-with-deepwalk-f5ac25900
           dw = DeepWalk(dimensions=2, walk_length=34*2 , walk_number=34)
           dw.fit(graphs_x)
           graph_embedding= dw.get_embedding()

CPU times: total: 312 ms
Wall time: 277 ms
```



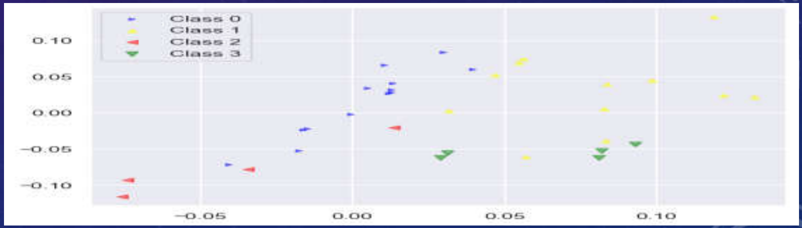
Execution Time of Gconv <<< DeepWalk DeepWalk is 10 times slower

Segregation of Nodes Embedding in ConvGNN is still Comparable to DeepWalk

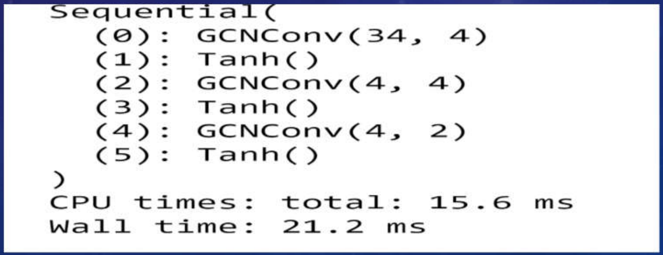
Uninitalized 3-ConvGNN (H=4, F=2, Tanh Activation, Fully-supervised, Non-Trained Weights, No Epochs I/P)



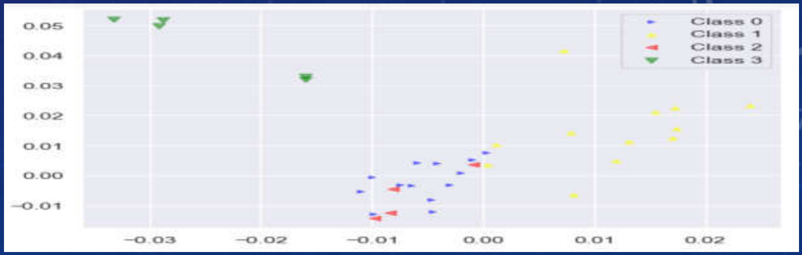
```
Sequential(
  (0): GCNConv(34, 4)
  (1): Tanh()
  (2): GCNConv(4, 4)
  (3): Tanh()
  (4): GCNConv(4, 2)
  (5): Tanh()
)
CPU times: total: 15.6 ms
Wall time: 14 ms
```



Uninitalized 3-ConvGNN (H=4, F=2, Tanh Activation, Semi-Supervised, Non-Trained Weights, No Epochs I/P , Training Label = 4 Nodes /34 Nodes)



```
Sequential(
  (0): GCNConv(34, 4)
  (1): Tanh()
  (2): GCNConv(4, 4)
  (3): Tanh()
  (4): GCNConv(4, 2)
  (5): Tanh()
)
CPU times: total: 15.6 ms
Wall time: 21.2 ms
```



CONCLUSION

REFERENCES