GRAPH NEURAL NETWORKS (GNN)

**Introduction to Graph Neural Networks and Graph Theory**

*Note: Graph Neural Networks are run basis graph theory. It is a discrete mathematics subject which has broader scope. Below are some of the concepts that are needed for understanding nuances of graph theory*

* Need to understand basics of Graph Theory like understanding of Graph, Node, Edge, and few terminologies on types of graphs (Bipartite, Complete, Incomplete graph).
* Node means vertex and Edge means connector between two nodes. Example like line between two points, where line is edge and points are vertices.
* Need to understand Adjacent and Incident Matrix computation in graph. Adjacent Matrix is square matrix of vertices (cross tab of which vertices are connected). Incident matrix is a matrix of vertices as rows and edges as columns. Both these matrices are used in the GNN models for mapping the neighbour nodes and assigning weights to relations
* Graph Neural Networks rely on Nodes and Edges
  + Node Features for predicting nodes output
  + Edge features can be used for predicting recommendation systems or complex social networks
* Bipartite graphs mean there can be two or more different types of nodes, that means same type of nodes will not have any links. This is mainly used in GNNs very heavily.
* GNNs work like Feed forward Neural networks, where the hidden layers are measured using weights and bias, and there will be a loss function (cross entropy for multiclass, sigmoid for binary class, MSE for regression models) for back propagation.

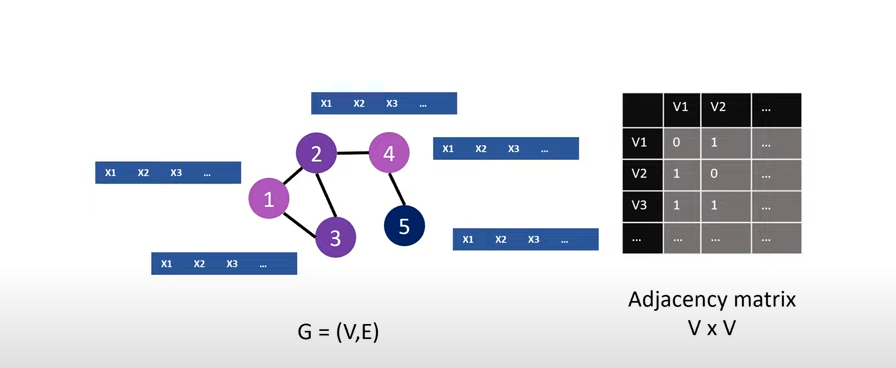


Figure showing how graph is converted into features and adjacency matrix.

**Types of Predictions that can be done using Graph Neural Networks (GNNs):**

*Node wise predictions:*

Here, the nodes are classified into different categories and features are mapped to the nodes. Now the unknown categorical nodes are predicted using the known categorical data of nodes

*Link Predictions:*

Example, in Social Networks, connections are recommended based on the strong links. Recommender systems recommend most suitable matches based on the link prediction

*Graph Prediction:*

Example, In molecule structure determination, the structure of the molecule is determined by the previous graphs

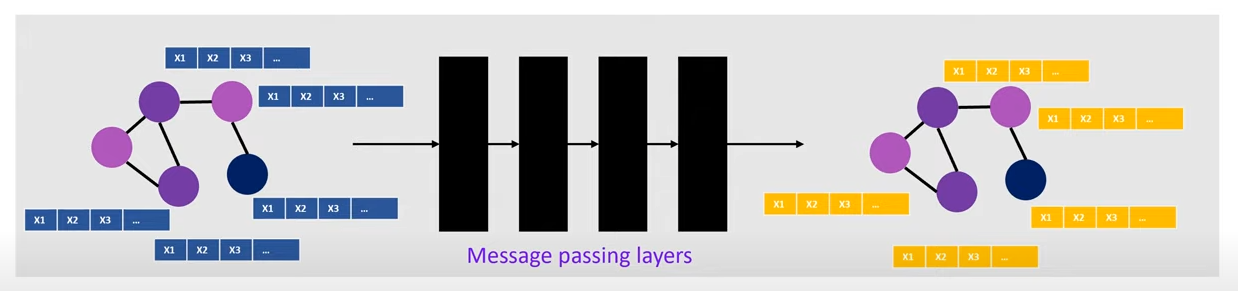
**Example Use cases:**

|  |  |
| --- | --- |
| **Domain** | **Usecase** |
| Pharmacy/Medicine | To determine the molecular structure of the compound |
| e-commerce | Recommendation systems to recommend products to users |
| Social Networks | To recommend connections, content etc., |
| Finance and Banking | Credit card transaction fraud, Identify Camouflage fraudsters |
| 3D-Games | Game meshes and rendering |

**How GNN Operate?**

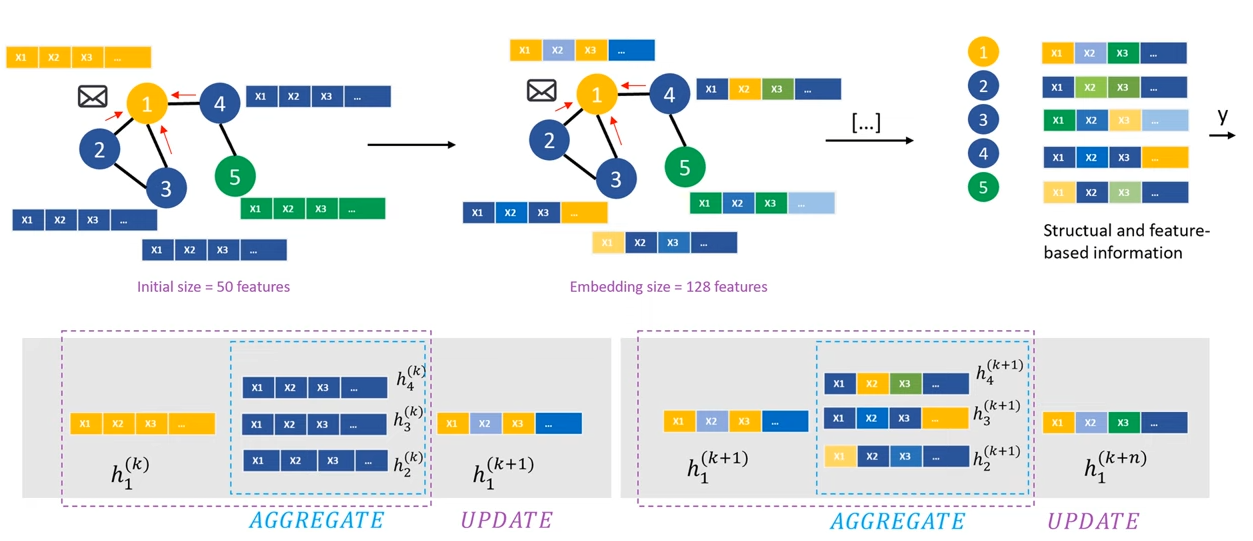
*Representation Learning:* Learning for neural networks suitable representation of graph data. That is the node features are represented such a way that the GNN model construct can be obtained from the nodes.

* What it means is each node of the graph will have properties which are considered as features.
* Now there is a concept of message passing, which means the neighbour nodes of the graph will pass some information over to the current node. Now, this is broken down as node embedding in the next hidden layers. The concept is more like CNN networks where we transform the pixel data into matrix transformations
* Now these properties/features are used to create message passing layers by creating node embedding.
* Node embeddings mean the information from one node is passed to another node

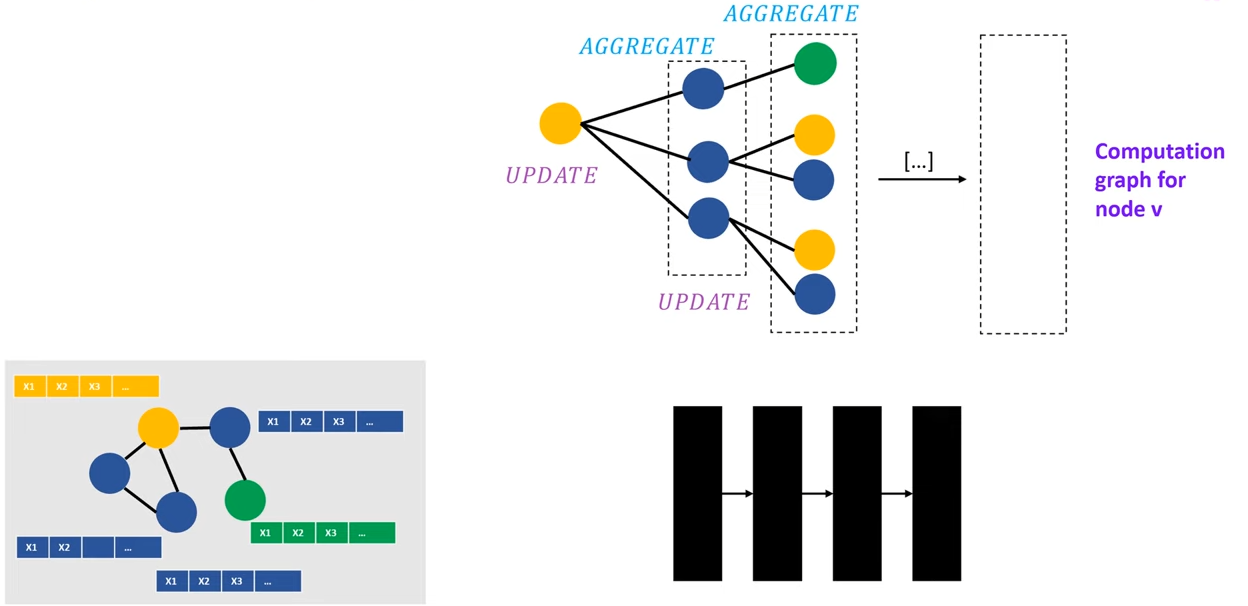


What are Message Passing Layers?

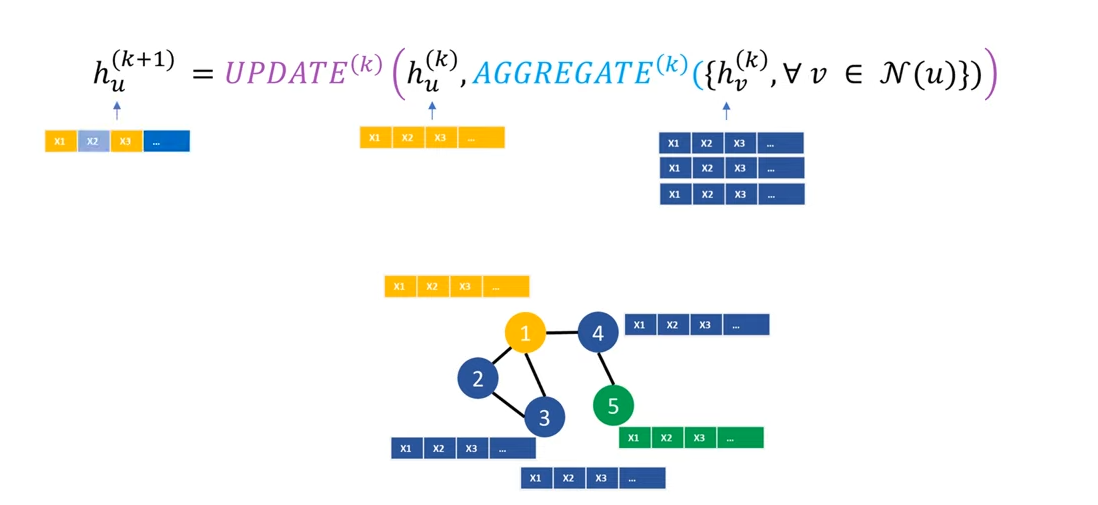
Each node passes message to another node and this process of information passing happens in each layer using aggregate and update mechanisms.



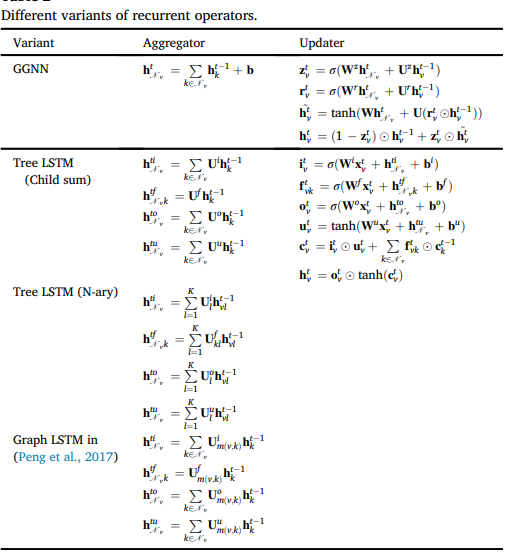
As we can see in the image below, there are two layers of message passing happened, and new embeddings are created at each node



Below Image, we are showing how the hidden layers get aggregated and updated basis previous.



Some Aggregate and Update functions:



**There are Supervisory Models and Unsupervisory models:**

1. Recurrent GNN and Convolutional GNN are supervisory models
2. Graph Auto Encoders (GAE) are un-supervisory models

**Libraries that are used most widely:**

Pytorch Geometric (most used)

Deep Graph Library

**Hyperparameters in GNN**

1. **Learning Rate**:
   * Controls how much the model's weights are adjusted with respect to the gradient during training. A small learning rate can lead to slower convergence, while a large learning rate might make the model overshoot the optimal solution.
2. **Number of Layers (Depth of the GNN)**:
   * No of message passing layers
3. **Hidden Units (Hidden Layer Size)**:
   * The number of hidden units in each layer of the network.
   * This determines the dimensionality of the node embeddings at each layer. Too few hidden units might limit the expressiveness of the model, while too many might lead to overfitting.
4. **Batch Size**:
   * The number of samples used in each iteration of training.
   * Smaller batch sizes often lead to noisier gradient updates but can help with generalization, while larger batch sizes provide smoother gradient estimates.
5. **Dropout Rate**:
   * Dropout is used to prevent overfitting by randomly setting a fraction of the neurons to zero during each training step.
   * The dropout rate specifies the probability of dropping a node in each layer.
6. **Activation Function**:
   * The function applied to the outputs of each layer to introduce non-linearity, such as ReLU (Rectified Linear Unit), LeakyReLU, or Sigmoid. This influences how well the model can learn complex patterns.
7. **Weight Initialization**:
   * Determines how the initial weights of the model are set before training begins. Proper initialization can improve training efficiency and help avoid issues like vanishing or exploding gradients.
8. **Graph-specific Parameters**:
   * **Edge Weight/Normalization**: Some GNNs use edge weights or normalization strategies to adjust how the graph information is aggregated during the message-passing process.
   * **Graph Sampling**: In large-scale graphs, methods like graph sampling (e.g., GraphSAGE) are used to select a subset of neighbors for each node to speed up computation.
9. **Regularization Parameters**:
   * **L2/L1 Regularization**: Regularization terms that penalize large weights to prevent overfitting.
   * **Graph Laplacian Regularization**: A technique that encourages smoothness in the learned node representations across the graph structure.
10. **Optimization Method**:
    * The choice of optimizer, such as Adam, SGD (Stochastic Gradient Descent), or RMSprop, and their respective parameters (e.g., momentum, beta values for Adam) affect the training process.
11. **Number of Epochs**:
    * The total number of passes over the training data. Training for too many epochs can lead to overfitting, while too few epochs may result in underfitting.

**Transactional Fraud Use case:**

Creating a graph data

1. Identifying Nodes, Edges, Node Features, Labels/Predictions
2. Homogenous Graph/Heterogenous graphs? Homogenous means all the node types are same. Example in below case if we only consider user as node. Heterogenous means node types are different, means user and transactions are marked as node.
3. If nodes are heterogenous, we need to separate the data and create model using heterogenous graphs

Torch\_geometric.dataloader library is used for graph data which has tensor data

Temporal Data or not? :

Temporal data means it’s a data split based on time intervals, as the data can be dynamic and changes based on time. Example, Node features can be updated, edge relationships can be updated etc.,

Transactional Fraud: Supervisory models

* User is a Node, and do node level prediction as whether user is fraud or not?
* User and Transaction is a node, use bipartite graphs, and predict whether user / transaction is fraudulent?

Node features example:

1. Hour of transaction
2. Country of user
3. State of user
4. State of transaction – Complete/Failed
5. Type of transaction
6. Currency of transaction
7. Previous fraud history of user like no of frauds
8. User topups count
9. No of past failed transactions
10. Country of transaction
11. Mean time of transactions
12. Velocity of transactions

Masking: You will mask the nodes which you are predicting and only used the nodes which have labels to train the loss function and optimize

Once you have trained the model, you can use the masked data and predict for the outcome

*Other important information*

* GNNs are GPU heavy models
* GNNs used tensor data as similar to Neural Networks

Appendix:

Sources:

[A Comprehensive Study on Graph Neural networks](https://arxiv.org/pdf/1901.00596)

[[2011.12193] xFraud: Explainable Fraud Transaction Detection](https://arxiv.org/abs/2011.12193)

[GitHub repository with all recent papers on Fraud models using GNNs](https://github.com/safe-graph/graph-fraud-detection-papers)

[Pytorch-Geometric installation guide](https://pytorch-geometric.readthedocs.io/en/latest/install/installation.html)

[Pytorch-Geometric Graph Neural Network Modules](https://pytorch-geometric.readthedocs.io/en/2.5.1/modules/nn.html)

Code for node level prediction:

[Example Code1:Using Cora dataset for node level predictions](https://colab.research.google.com/drive/1LJir3T6M6Omc2Vn2GV2cDW_GV2YfI53_?usp=sharing#scrollTo=lcffG3rQy79R)

[Example Code2: Molecule Solute problem prediction](https://colab.research.google.com/drive/14OvFnAXggxB8vM4e8vSURUp1TaKnovzX?usp=sharing#scrollTo=eqWR0j_kIx67)

[Youtube Series of GNNs](https://www.youtube.com/watch?v=ABCGCf8cJOE&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=3)