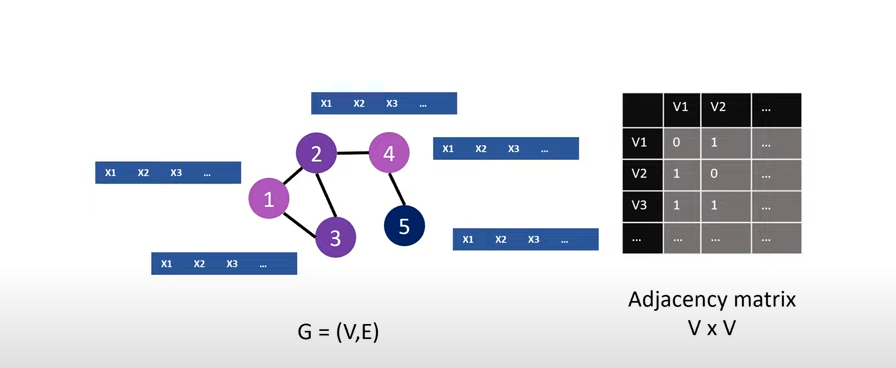
GRAPH NEURAL NETWORKS

Graph Neural Networks are run basis graph theory

* Need basics of Graph Theory
* Adjacent and Incident Matrix
* Nodes and Edges
  + Node Features
* Bipartite graphs
* They work similar to Feed forward Neural networks



**Data Properties:**

1. Size Independent
2. Permutation Invariance
3. Non-Euclidean Space

**Use cases:**

Pharmacy 🡪 To determine the Molecular structure of a compound

Ecommerce 🡪 In Recommender Systems, to determine the ideal products to recommend

Social Networks 🡪 People as Nodes, networks, to determine relationships

3D Games 🡪 Games Meshes and rendering

**Types of Predictions:**

Node Prediction 🡪 Relationship prediction models, is this person trying to do fraud? Is this person a fraud?

Link Prediction 🡪 Recommender Systems

Graph Prediction 🡪 Molecular Structures

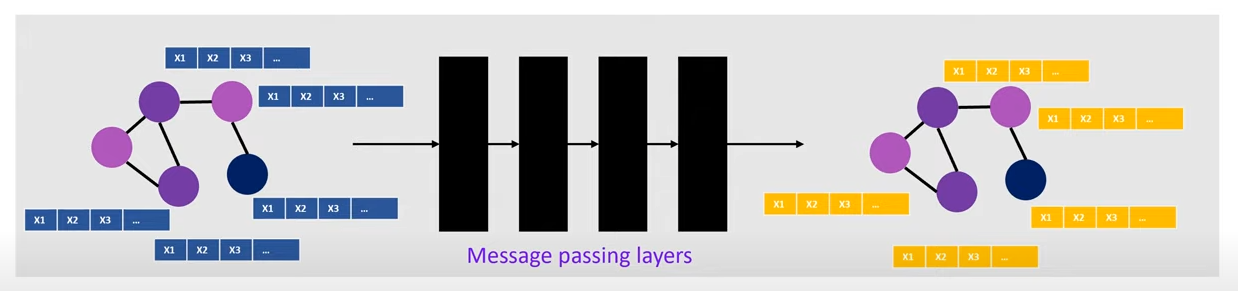
Cannot use Adjacency matrix in feed forward neural networks as the matrix is sensitive to permutation / change in input

Isomorphic graph data should be identified in the model

**How GNN Operate?**

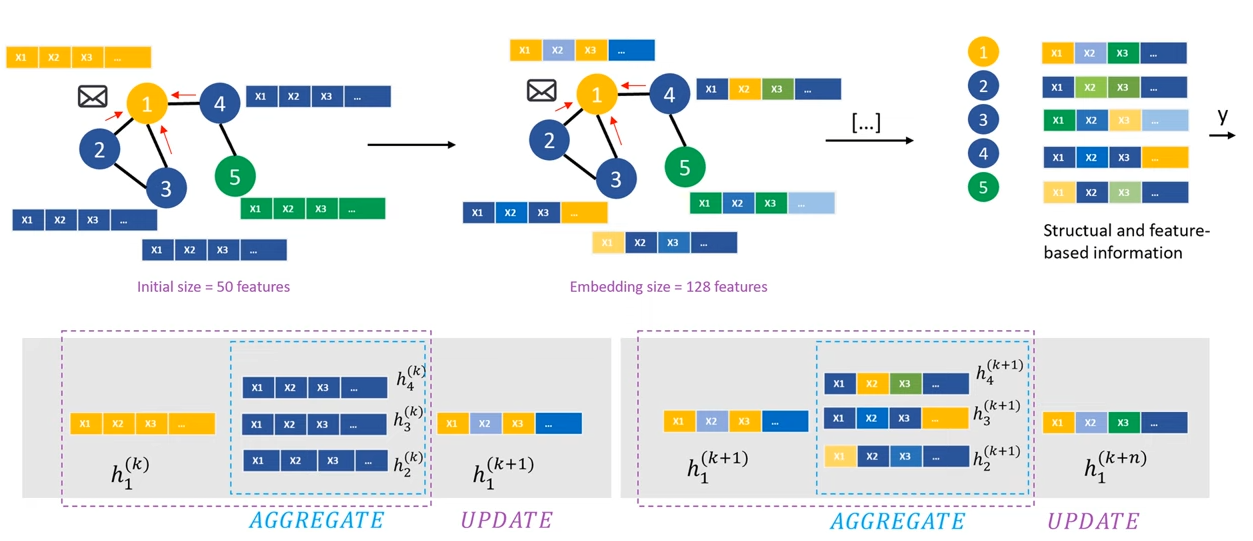
Representation Learning: Learning for neural networks suitable representation of graph data

* What it means is each node of the graph will have properties.
* Now these properties are used to create message passing layers by creating node embedding.
* Node embeddings mean the information from one node is passed to another node

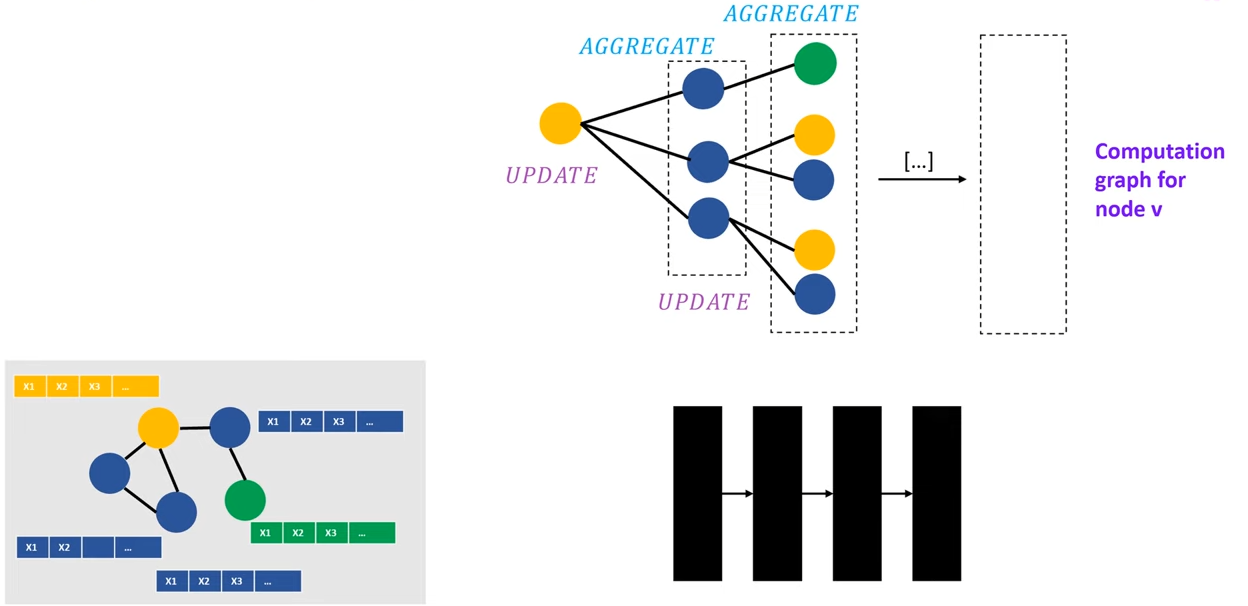


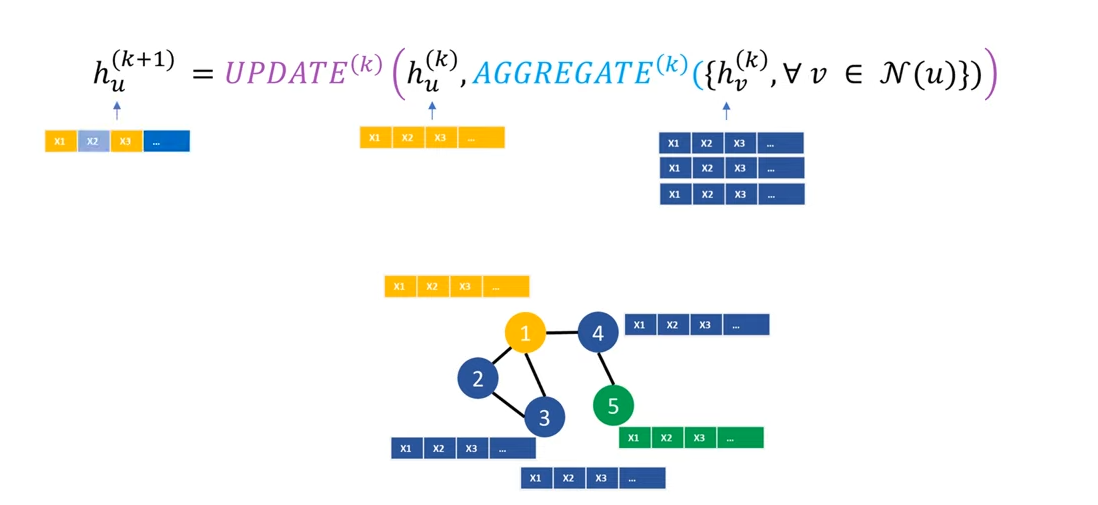
Message Passing Layers

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





**Libraries that are used most widely:**

Pytorch Geometric

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. **Learning Rate Scheduler**:
   * Defines how the learning rate changes during training. Common strategies include reducing the learning rate after a certain number of epochs or using adaptive methods like learning rate annealing.
9. **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.
10. **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.
11. **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.
12. **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.

**Libraries:**

Pytorch Geometry

Deep Graph Library

**Transactional Fraud Use cases:**

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

Source: <https://www.youtube.com/watch?v=ABCGCf8cJOE&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=3>

<https://arxiv.org/pdf/1901.00596>

<https://pytorch-geometric.readthedocs.io/en/latest/install/installation.html>

<https://github.com/safe-graph/graph-fraud-detection-papers>