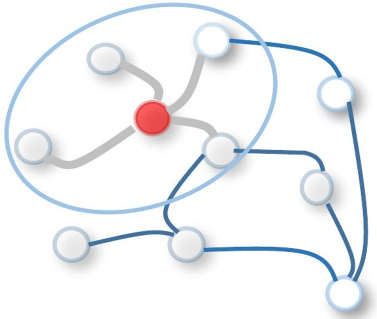
Graph Neural Networks

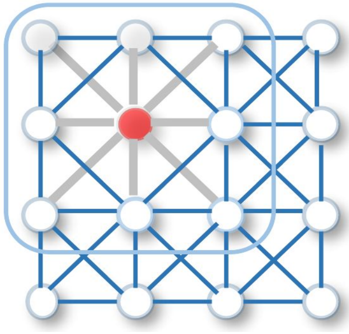
Overview and Some Interesting Recent Works

GNN - What is it?

- Framework to obtain Representation for graphs
- Node feature: Recursively transform & aggregate features of neighbors



Graph

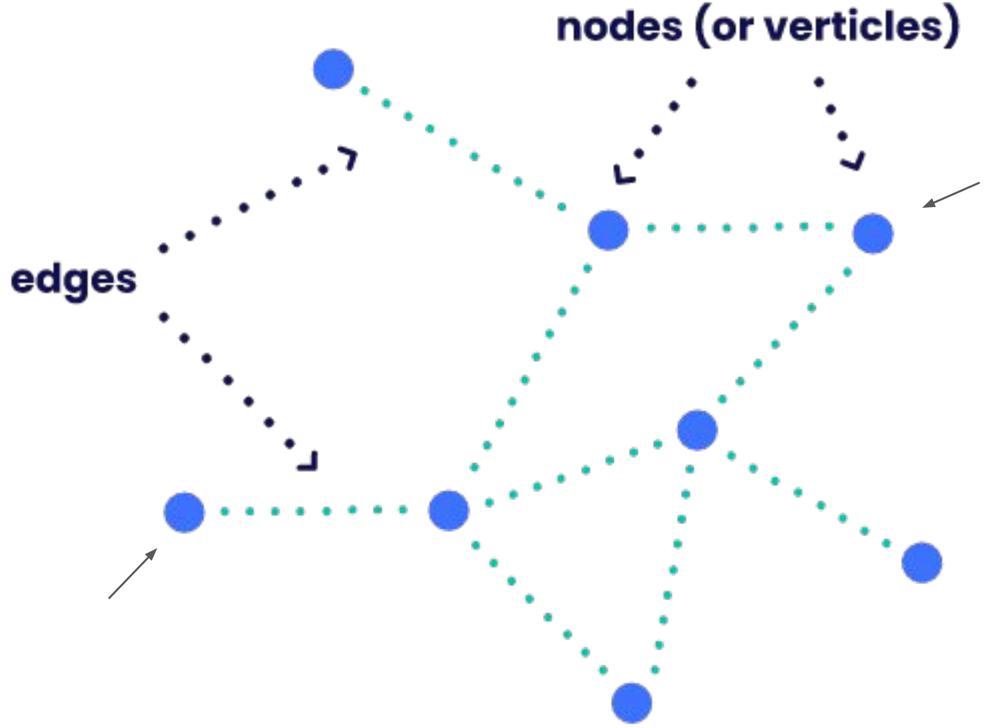


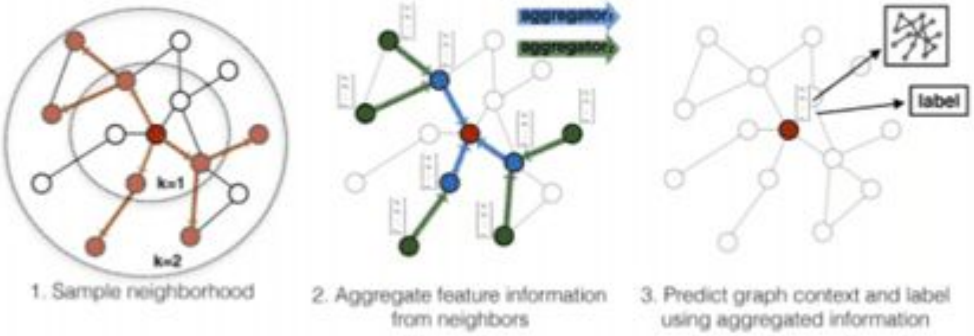
Image

- Why care for GNNs? Applied in many real-world apps. - NLP, Vision
  - Tasks such as Node classification, Link prediction etc. for social net., protein graphs etc.

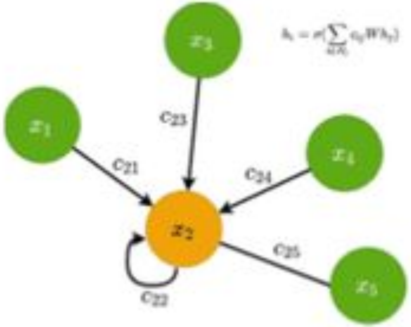
Image Src: [1] Z. Wu et al.

What are Graphs

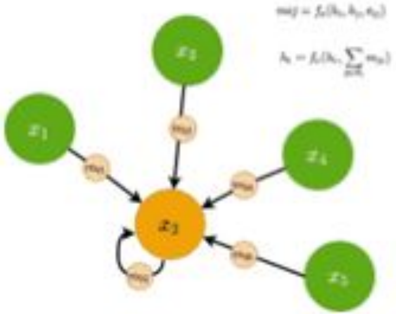




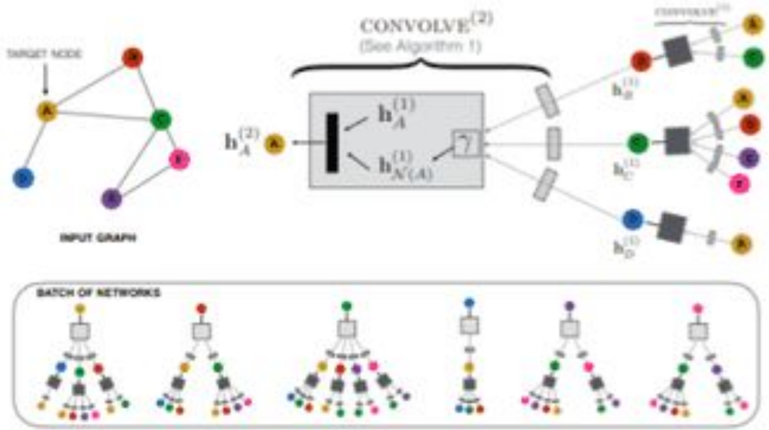
k=1
k=2
1. Sample neighborhood
aggregator
aggregator
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information



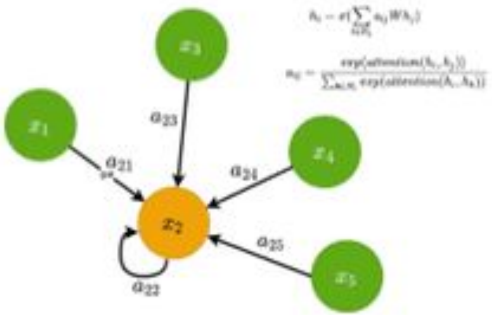
x_3
x_1
x_4
x_2
x_5
c_{21}
c_{22}
c_{23}
c_{24}
c_{25}



x_1
x_2
x_3
x_4
x_5



TARGET NODE
INPUT GRAPH
CONVOLVE$^{(2)}$
(See Algorithm 1)
$h_A^{(2)}$
$h_A^{(1)}$
$h_{N(A)}^{(1)}$
$h_B^{(1)}$
$h_C^{(1)}$
$h_D^{(1)}$
BATCH OF NETWORKS



x_3
x_1
x_4
x_2
x_5
a_{21}
a_{22}
a_{23}
a_{24}
a_{25}

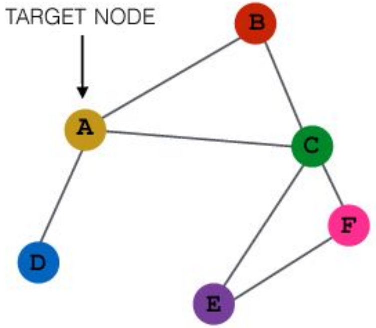
GNN (contd.)

B1 = B + W0.(A+C)
C1 = C + W0.(A+B+E+F)
D1 = D + W0.(A)

N(A)=W1.(B1+C1+D1)

A = A1 + N(A)

- Variants:
  - Recurrent GNNs, Convolutional GNNs , Graph Autoencoders etc.
- Here, we consider, a broader definition
  - Any recursive neighborhood aggregation (or message passing)
  - After k-iterations, a node captures structural info. of k-hop neighborhood



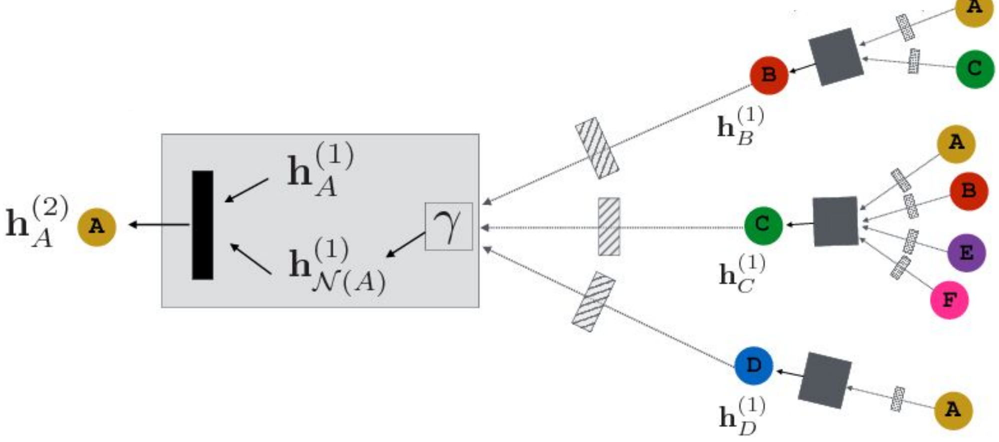
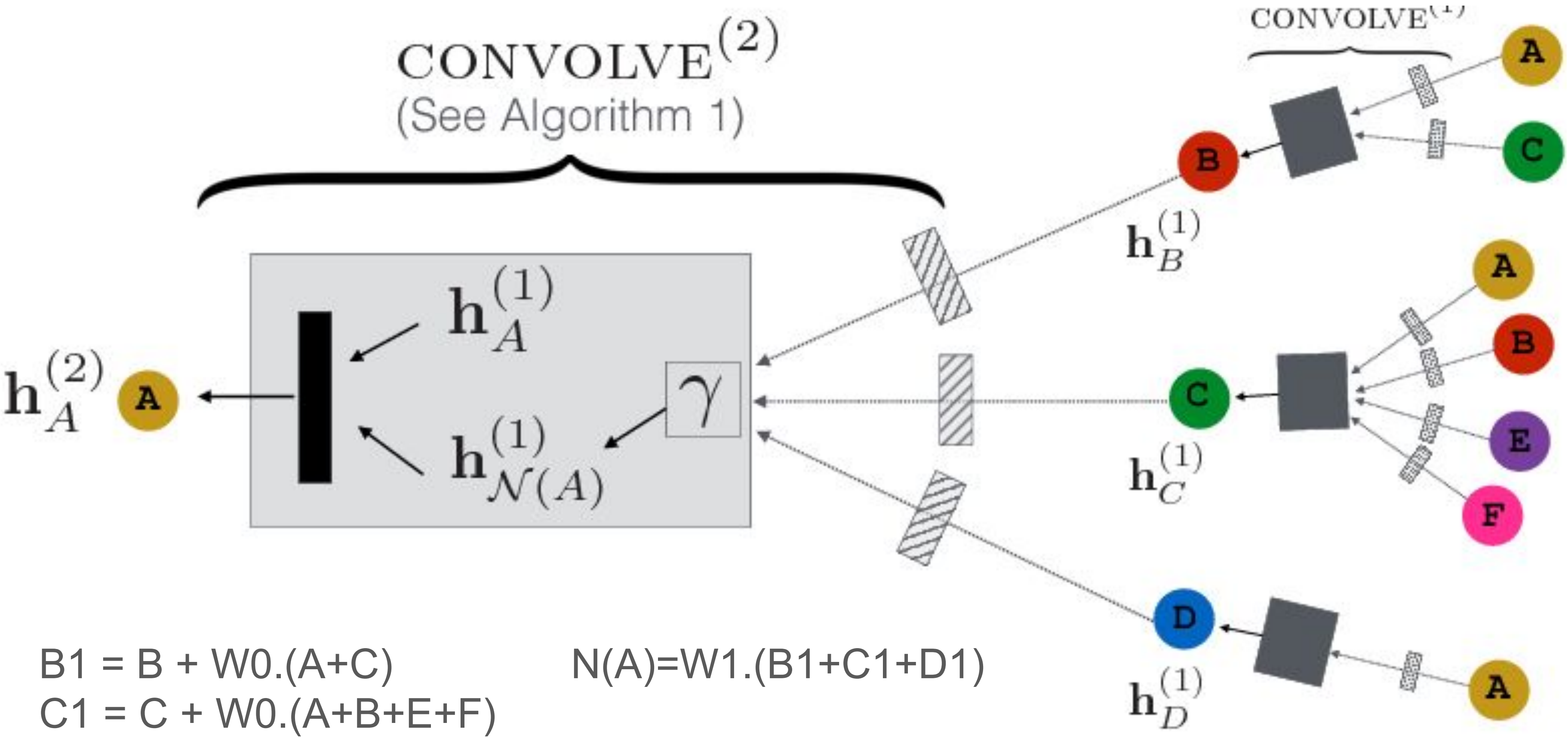


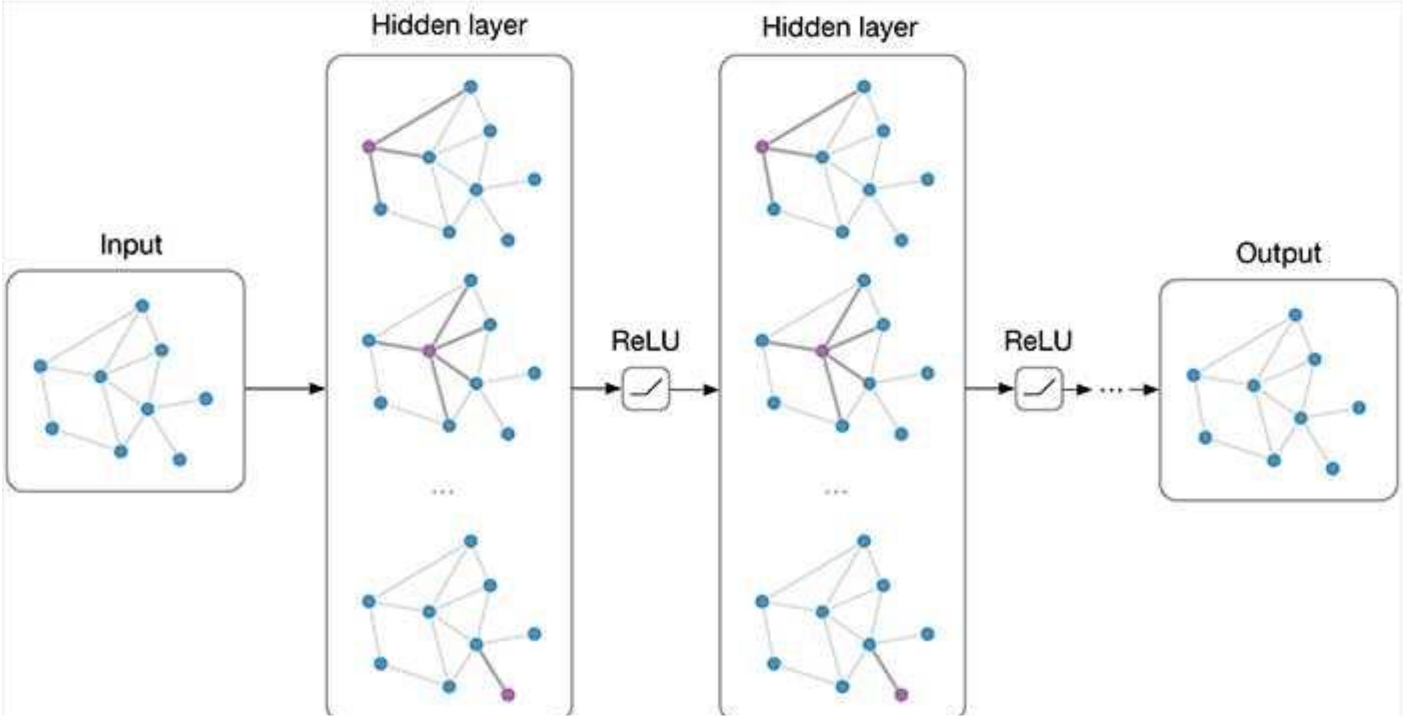
Image Src: [2] Jure Leskovec WWW



B1 = B + W0.(A+C)
C1 = C + W0.(A+B+E+F)
D1 = D + W0.(A)

N(A)=W1.(B1+C1+D1)

A = A1 + N(A)


Hidden layer
Hidden layer
Input
ReLU
ReLU
Output

Notation

$$\begin{aligned} G &= (V, E) \text{ Graph} \\ h_v &: \quad \text{representation vec. of node } v \in V \end{aligned}$$

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)$$

Aggregate all neighbours of a node

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

Combine node’s representation
with that of its neighbours

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)$$

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

~~Message Passing~~ Update in its generality

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)$$

Aggregate all neighbours

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

Combine Node and its ngbr's Aggregate

$$\mathbf{x}_i^{(k)} = \gamma^{(k)}\left(\mathbf{x}_i^{(k-1)}, \square_{j\in\mathcal{N}(i)}\, \phi^{(k)}\left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i}\right)\right),$$

Combine function

Aggregate function

Formulate a message that will pass through the edge connecting x_i and x_j

$$\mathbf{x}_i^{(k)} = \boxed{\gamma^{(k)}\left(\mathbf{x}_i^{(k-1)}, \boxed{\square_{j\in\mathcal{N}(i)}\, \underbrace{\phi^{(k)}\left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i}\right)}_{\text{Formulate a message that will pass through the edge connecting x\_i and x\_j}}}\right)},$$

Combine function

Aggregate function

Convolution Layers for Message-Passing

Some Popular Architectures

GCN - Classic GNN

Graph Convolutional Operator

$$\mathbf{x}'_i = \mathbf{\Theta} \sum_{j} \frac{1}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j$$

In words: A node’s representation = sum of its neighbors, normalized by their degrees

Preprocessing: (A+I) - node is neighbor of itself

Message (for each edge j → i): normalized (xj)

Aggregate: \sum (all messages)

Combine: \theta * aggregate

Parameter: \theta R^{NxM}, di, dj are degrees of nodes i, j

Real world example - Coauthor graph

Say we have co-author network (cora, citeseer),

- **Nodes**: Authors who publish papers
- **Edges**: Between two authors if they have co-authored a paper

Task: to classify authors into 7 classes, based on primary research area of the author (like ML, Databases, Security, Networks etc.)

GCN can be helpful - Authors are largely expected to be coauthors with researchers from the same field.

Why Normalize degrees?

- Additive bias like ML/AI might have relatively more authors

GraphConv - another classic method

Graph Convolutional Operator

$$\mathbf{x}'_i = \mathbf{\Theta}_1 \mathbf{x}_i + \mathbf{\Theta}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j$$

In words: Very similar to previous GCN, with one more additional freedom, both self and neighbors are weighted differently.

Message (for each edge j → i): weighted (xj)

Aggregate: \sum (all messages)

Combine: \theta_1 * self + \theta_2 * aggregate

Parameter: \theta_1, \theta_2 R^{NxM}

Real world example - Reddit Data

Reddit graph has posts from top 50 popular subreddits

- **Node**: posts, init-node-features: avg. word embedding
- **Edges**: between two posts if they share same user who has commented on both the posts.

Task: To classify the posts, according to their subreddits.

GraphConv - calculated node embeddings by weighing self and neighbors differently. It performed best.

Note: GraphConv is a strict generalization of GCN.

GAT - Graph Attention Network

Earlier weights were given (and fixed), now they are learned.

Graph Attention Operator

$$\mathbf{x}'_i = \alpha_{i,i}\mathbf{\Theta}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}\mathbf{x}_j,$$

attn. weights \alpha_ij calculated as

$$\alpha_{i,j} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^{\top}[\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_j]\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^{\top}[\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_k]\right)\right)}.$$

Message: xj

Aggregate: calculate \alpha ; \sum (messages * alpha)

Combine: \theta [self + aggregate]

Parameter: theta = R^{NxM}

Real world example: Protein-Protein Interaction

PPI graph has human-tissue interactions. It is densely interconnected.

- **Node**: human tissues, node-feature: gene signatures
- **Edges**: between two tissues if they interact in some way

Task: To assign multi-class labels from 121 classes

GATs could learn to focus on the relevant connections and ignore the spurious (less important) ones.

RelConv - Relational Network

RCN Operator

$$\mathbf{x}'_i = \mathbf{\Theta}_{\mathrm{root}} \cdot \mathbf{x}_i + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|} \mathbf{\Theta}_r \cdot \mathbf{x}_j,$$

In words: Same as Graph conv, but edges are labeled, so neighbors belonging to different relation types learn separate \theta_r.

Message (for each edge j → i): theta_r (xj)

Aggregate: \sum (all messages)

Combine: theta*self + aggregate

Parameter: \theta_r for each relation type 'r'.

Real world example: Product Ratings

User-product ratings graph with labeled edges of ratings R ={1,2,3,4,5}

- **Node**: users and products (bipartite)
- **Edges**: between user and product if user has rated the product, with label R

Task: To predict rating for a (user-product) pair

First RelConv finds node-embeddings for both products and users.

RelConv learns separate W_r for each rating score R.

- There is importance in treating each rating differently, 1⭐ versus 5⭐.

GINConv - Graph Isomorphism Network

GIN Operator

$$\mathbf{x}'_i = h_{\mathbf{\Theta}} \left((1 + \epsilon) \cdot \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \right)$$

h_\theta = is a MLP

MLP = \sigma(Wx+b)

One of the most powerful GNN models

In words: Just like the classic GraphConv, with one additional MLP at the Combine step.

Message (for each edge j → i): (xj)

Aggregate: \sum (all messages)

Combine: MLP [(1 + \epsilon) * self + aggregate]

Parameter: MLP that is learned

Real world example: Graph Classification

Training has several graphs, and labels are assigned to entire graph

- **Node**: different as per context: IMDB, Collab, Proteins etc..
- **Edges**: as per graph's context
- **Preprocessing**: all node and edge features are removed.

Task: To predict graph label only using its structure

GIN can map entire graphs to embeddings, which give best accuracy.

Sample Code for node classification

Quite easy to use pre-existing networks. (similar to Pytorch modules)

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = SAGEConv(data.num_node_features, 16)
        self.conv2 = SAGEConv(16, data.target_num_classes)

    def forward(self, data):
        x = self.conv1(x, edge_index)
        x = torch.nn.functional.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

dim= (input x 16)
dim = (16 x #classes)

2 Layer GNN module

Notice explicit non-linearity after node-aggregation

Forward propagation method