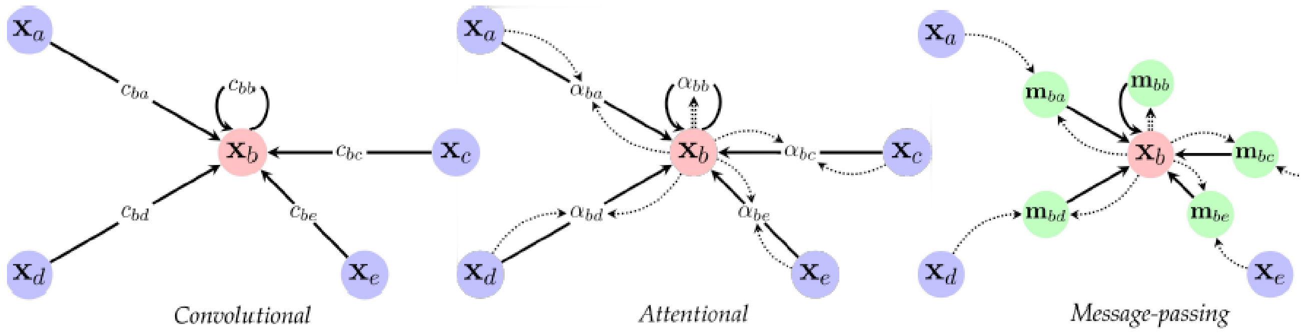
GNN Architectures



$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$

e.g. GraphSAGE, GCN, SGC

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$

e.g. MoNet, GAT, GATv2

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

e.g. IN, MPNN, GraphNet

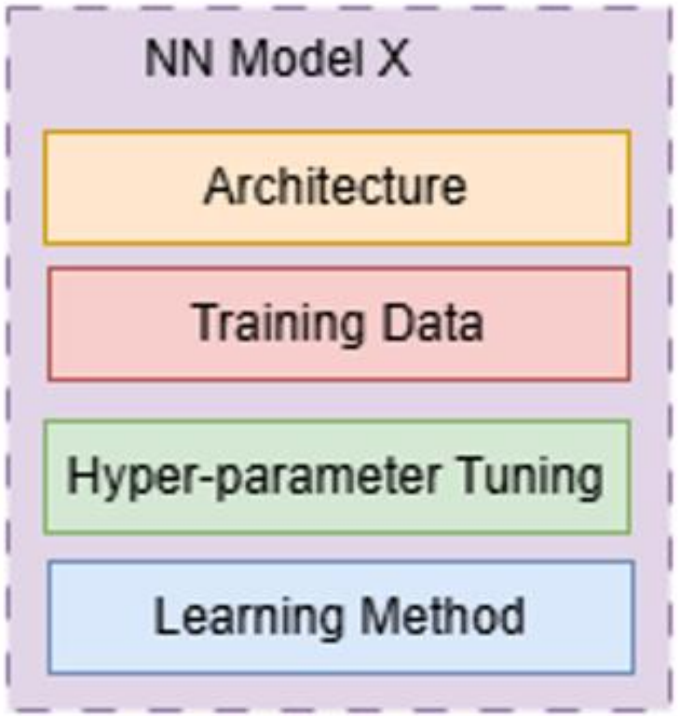
BRAIN LABS
Brain-Inspired AI & Neuroinformatics Research Group

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

ICARC
International Conference on Advanced Research in Computing

A Quick Word....

- Graph Neural Network (GNN) require graph structured data.
- If the data is not graph structured, data need to be converted to a graph structure.
- Always understand the data before implementations.
    - Visualize the graph
    - Summary statistics
- Lots of standard architectures...choose wisely..!
    - Expressivity
    - Spatio-temporal
    - Temporal
- Hyper-parameter tuning is same as in CNN and FFNN models
- Learning methods are same as used in CNN and FFNN models



Frameworks and Libraries

- PyTorch Geometric (PyG) and Deep Graph Library (DGL) are the two main GNN libraries used by researchers and industry
- PyG support only PyTorch while DGL support both the PyTorch and TensorFlow frameworks.
- NetworkX is a library used for network analysis, visualization and manipulation.

PyTorch Geometric

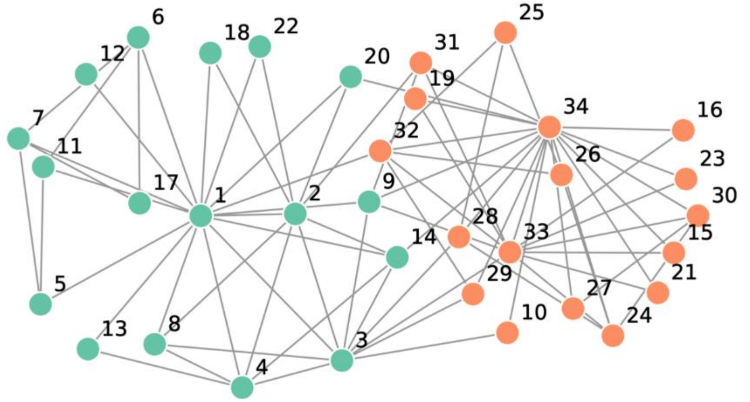
NetworkX

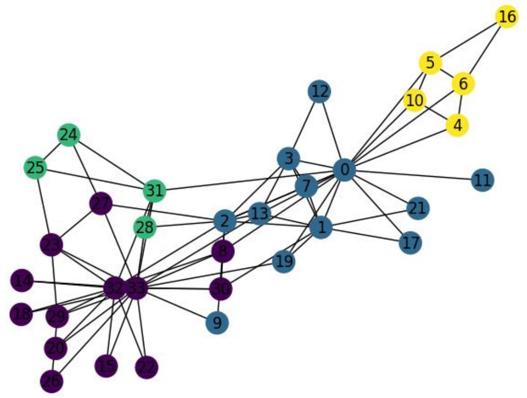
DGL

Spektral

Iris Dataset of GNN :Zachary Karate Club

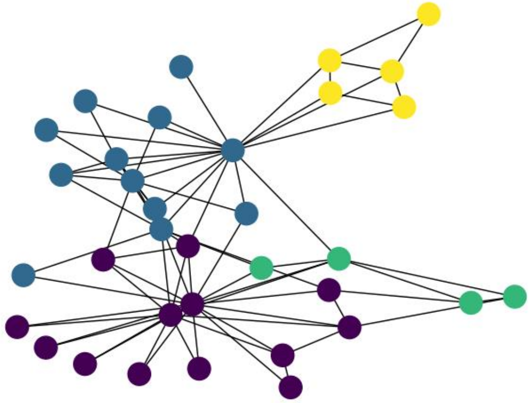
- Zachary's Karate Club network is a **social network dataset** that represents the relationships between members of a karate club at a U.S. university in the 1970s.
- The network is an **undirected graph** with **34 nodes** (each representing a club member).
- The dataset captures the **friendship ties (edges)** between members based on observed interactions resulting in 78 edges.
- Club **split into two factions** due to a conflict between the **club instructor** (node 1) and the **club administrator** (node 34).
- This division is observable through network analysis thus this is ideal dataset for GNNs as well.
- Due to small size and low complexity, this dataset can be considered as the "hello world" dataset of GNN research and can be compared to Iris dataset that play a similar role with classical ML research.
- Feature vector contains 34 values and altogether 4 classes.

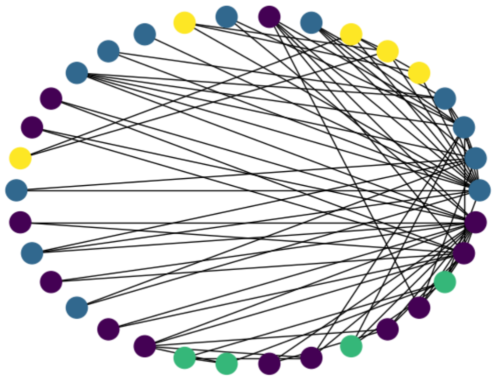


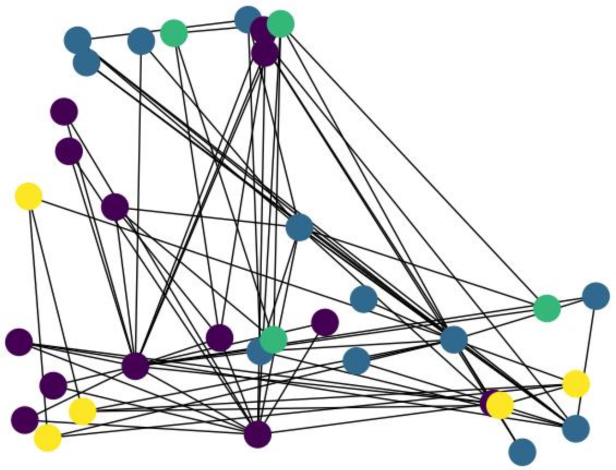


Visualization

- NetworkX is used.
- PyG graph should be first converted to a network graph
- Use the draw method!
- Different drawing layouts can be selected
- If none given, spring layout is used
- Different layouts
    - Random
    - Spring
    - Circular
    - Planar - only if possible
    - Bipartite - only if possible







For GNN: PyTorch >>> TensorFlow

- PyG only supports PyTorch
- Two main differences between TensorFlow and PyTorch
    - Model need to be defined as a class (similar to subclassing approach in TF)
    - Training loop need to be explicitly written (no method like model.fit)

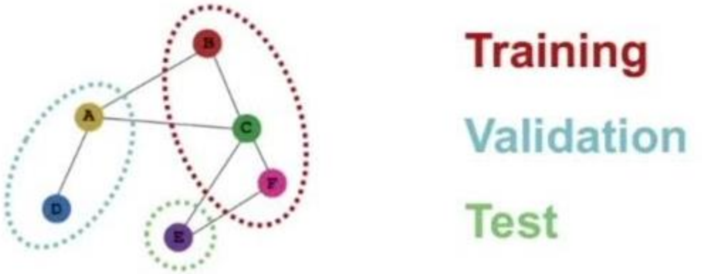
```python
class GCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        torch.manual_seed(1234)
        self.conv1 = GCNConv(dataset.num_features, 4)
        self.conv2 = GCNConv(4, 4)
        self.conv3 = GCNConv(4, 2)
        self.classifier = Linear(2, dataset.num_classes)

    def forward(self, x, edge_index):
        h = self.conv1(x, edge_index)
        h = h.tanh()
        h = self.conv2(h, edge_index)
        h = h.tanh()
        h = self.conv3(h, edge_index)
        h = h.tanh()  # Final GNN embedding space.

        # Apply a final (linear) classifier.
        out = self.classifier(h)

        return out, h

model = GCN()
```

```python
def train(data):
    optimizer.zero_grad()
    out, h = model(data.x, data.edge_index)
    loss = criterion(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()
    return loss, h
```

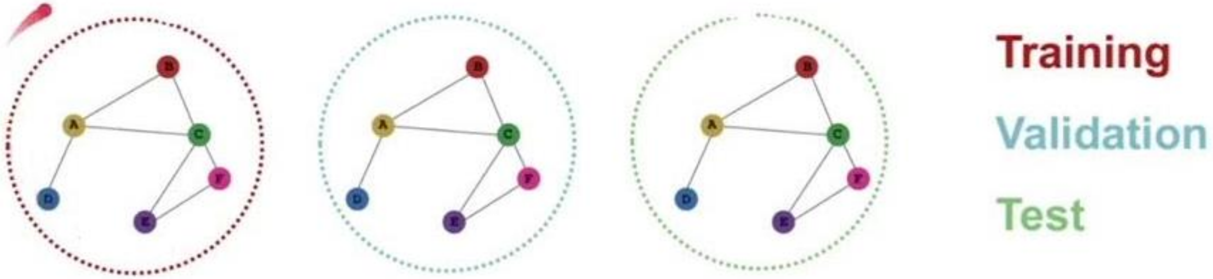
```python
for epoch in range(401):
    loss, h = train(data)
    losses.append(loss)
    print(f"Epoch: {epoch}\tLoss: {loss:4f}")
```

Another important point…

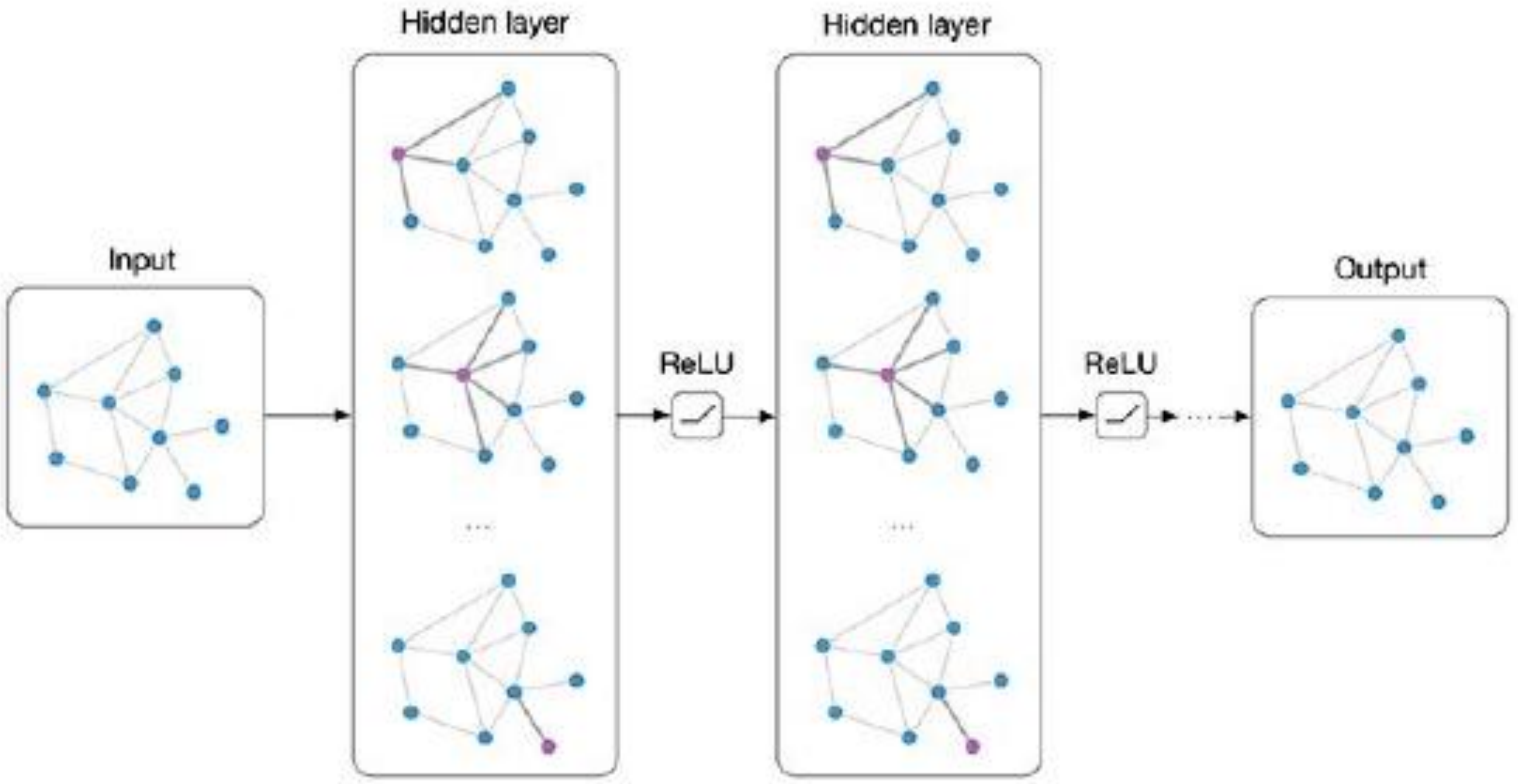
- **Transductive node classification**



- **Inductive node classification**



GNN Architectures



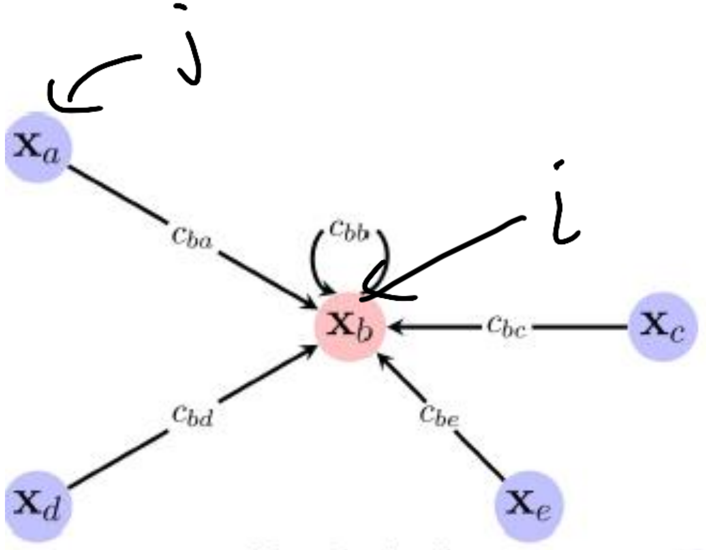
Graph Convolutional Network (GCN)

- MPGNN equation

$W_{self} = W_{neigh}$

$$h_u^{(k)} = \sigma\left(W_{self}^{(k)} h_u^{(k-1)} + W_{neigh}^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} + b^{(k)}\right) \rightarrow \begin{bmatrix}1\\0\\1\end{bmatrix}$$

- GCN use weight sharing thus there is only one weight matrix
- Some GCN normalizes the aggregated messages by degree.
- Other GCNs use re-normalization method.



Convolutional

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} h_j^{(l)} W^{(l)}\right),$$

$\times \rightarrow c_{ij} = d_i$

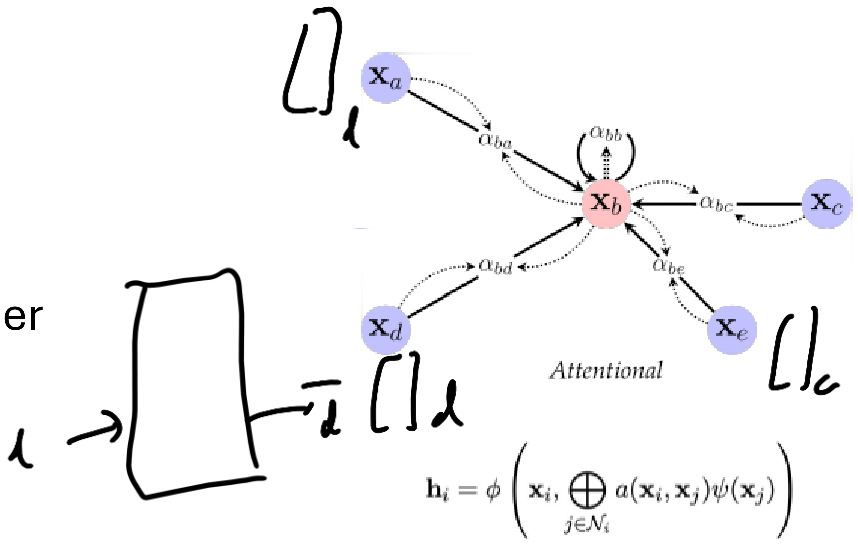
$\checkmark \rightarrow c_{ij} = \sqrt{d_i \times d_j}$

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij} \psi(\mathbf{x}_j)\right)$$

$$\sum \frac{1}{\sqrt{d_i d_j}} h_j \underline{W}$$

[1] Kipf, T.N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Graph Attention Network (GAT)



- Attention mechanism has shown promising results with Transformer architectures in many areas.

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

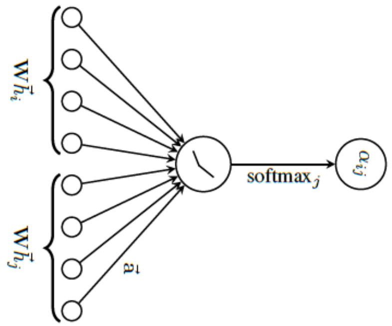
- Attention coefficient is obtained (e_ij) using a(.) which indicate the importance of node j's features to node i's.
- Global attention looses structural information so neighborhood structure is used with attention (i.e., local attention).

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}.$$

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right).$$

$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$



[2] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. and Bengio, Y., 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.