ETH zürich

BIOMEDICAL INFORMATICS

GNN Pipeline Construction & Training

Institute for Machine Learning, Department of Computer Science

Outline for Today

1st slot (45 min) - Lecture:

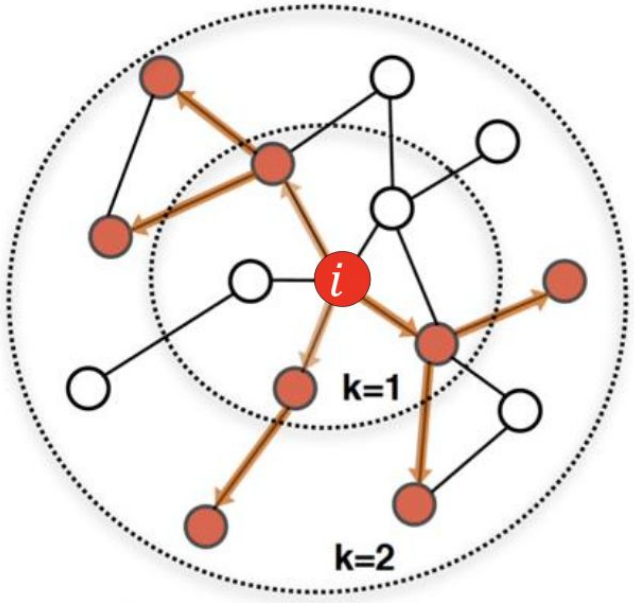
1. Recap
2. Goals
3. Over-Smoothing Problem
4. Training Pipeline Construction
5. Summary & Take-Home Messages

2nd slot (45 min) - Project 1 distribution.

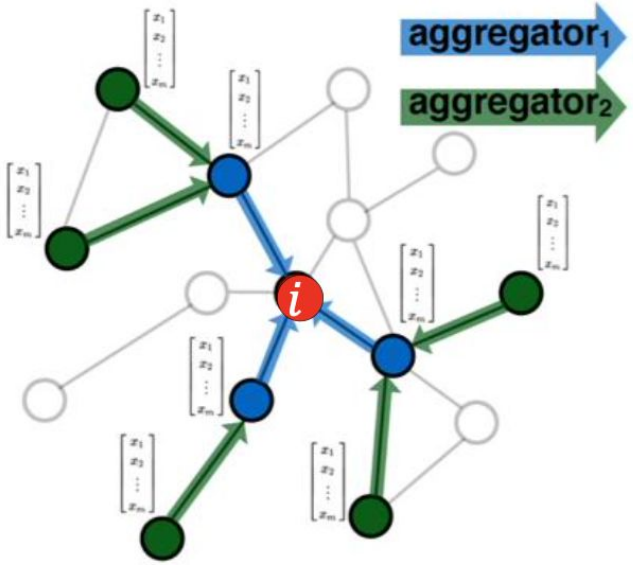
Recap: GNN Intro

Recap: Graph Neural Networks

Idea: node's neighbourhood defines a computation graph.



Determine node computation graph

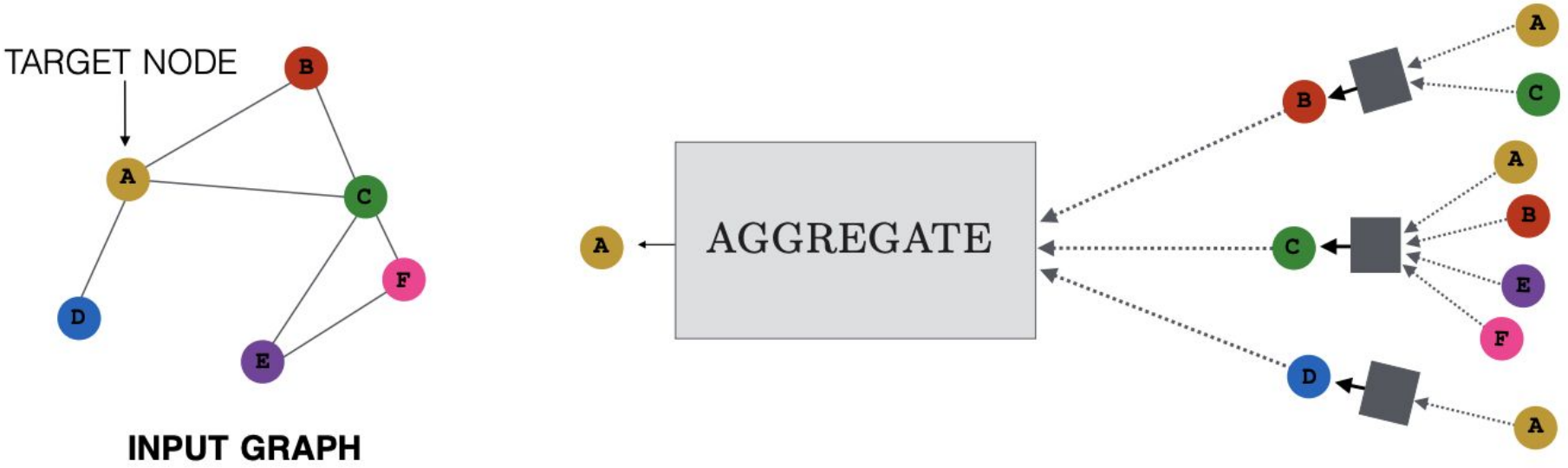


Propagate & transform the information

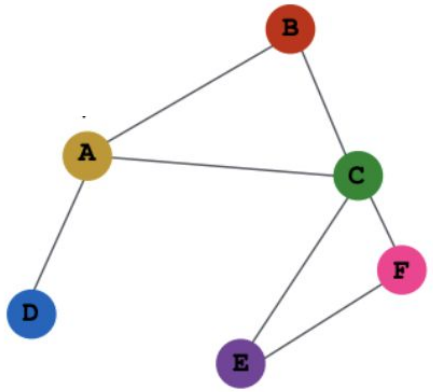
Credit: [1]

Recap: Neural Message Passing

Key idea: generate node embeddings based on local network neighbourhood.



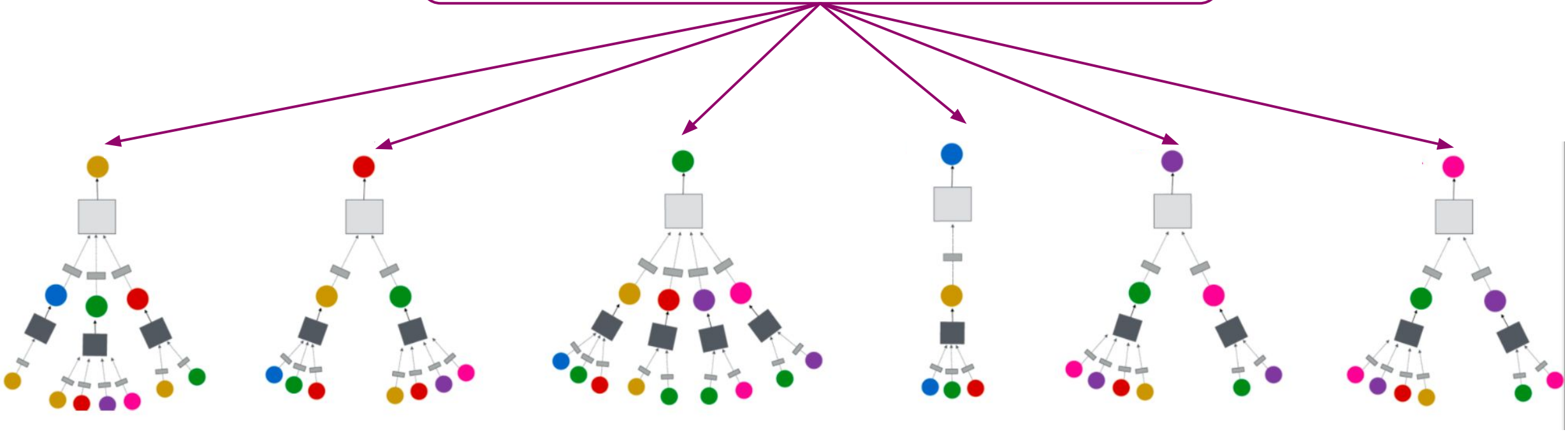
Recap: Neural Message Passing



INPUT GRAPH

Network neighbourhood defines a computational graph.

Every node defines a computational graph.



Recap: Neural Message Passing

1. **AGGREGATE** function takes as input the set of embeddings of the nodes in v's graph neighborhood $\mathcal{N}(v)$ and generates the message based on this aggregated information.

$$h_v^0 = x_v, \forall v \in V$$

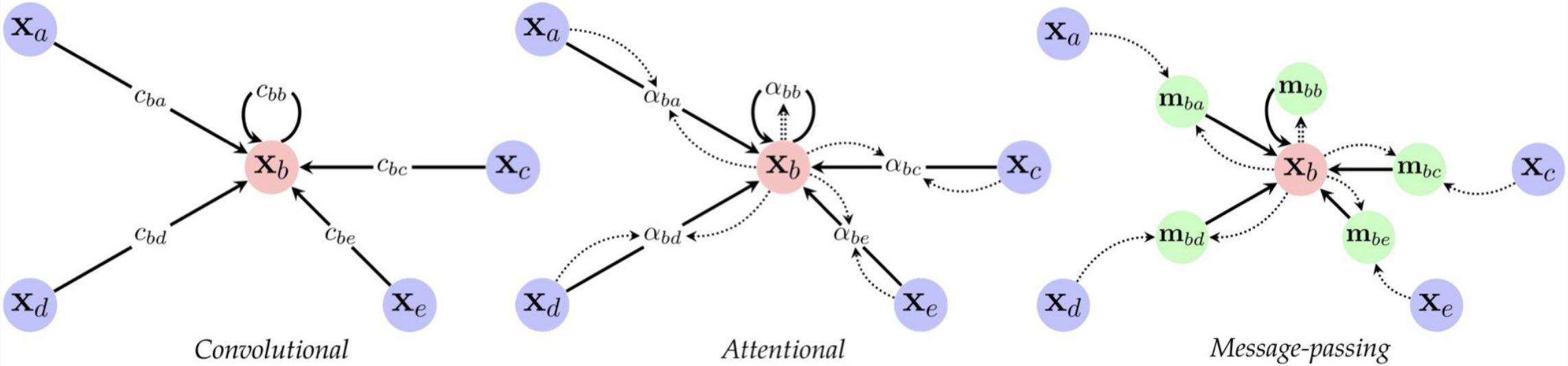
For each step (layer) $k = 1, ..., K$:

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

2. **UPDATE** function combines this message with previous node's v embedding to create the new embedding ($k = 1, ..., K$):

$$h_v^k = \text{UPDATE}(h_v^{k-1}, m_{\mathcal{N}(v)}^{k-1})$$

Recap: Zoo of 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

Credit: [3]

Notes on Normalization

Recap:

$$h_v^k = f^k\left(W^k \frac{\sum_{u \in \mathcal{N}(v)} h_u^{k-1}}{|\mathcal{N}(v)|} + B^k h_v^{k-1}\right), \forall v \in V$$

Normalization Option 1 - simply average

$$h_v^k = f^k\left(W^k \sum_{u \in \mathcal{N}(v)} \frac{h_u^{k-1}}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}} + B^k h_v^{k-1}\right), \forall v \in V$$

Normalization Option 2 – symmetric normalization

Option 2 – in some tasks information from very high-degree nodes may not be very useful for inferring membership.

Goals for Today's Lecture

1. How to stacking more GNN layers successfully
   a. What is an over-smoothing problem.
   b. How to overcome it designing layer connectivity.
2. Training Pipeline Construction
   a. Different Prediction Heads.
   b. Losses and Metrics.
   c. How to split the data.

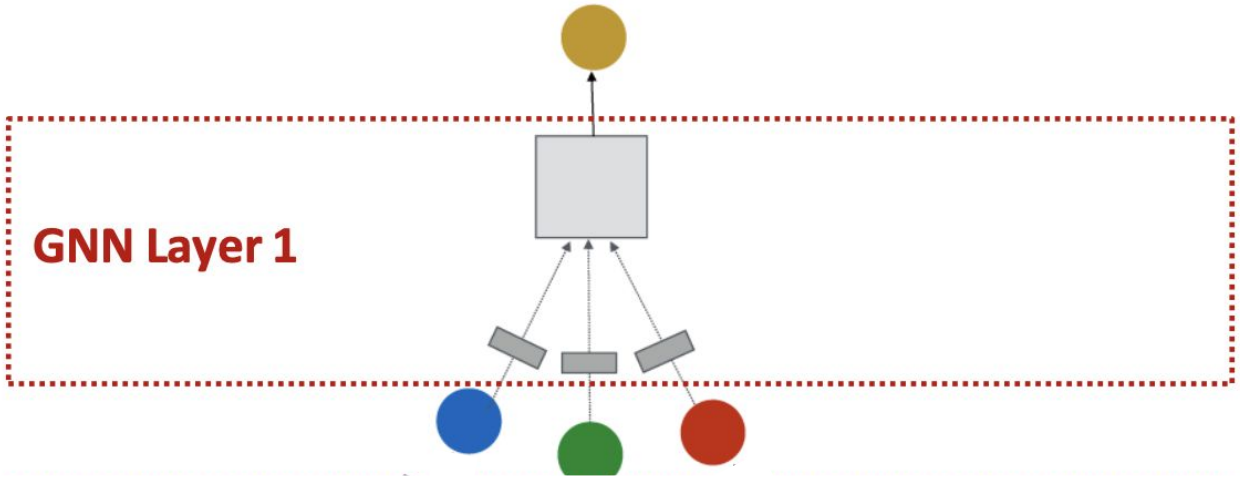
After this Lecture - we are finally ready to train!

Deep Learning with GNNs

Neural Message Passing: Summary

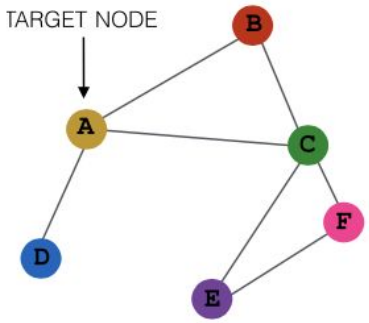
Two-step process:

1. **Aggregate** messages from the neighbours
2. **Update** the node features.



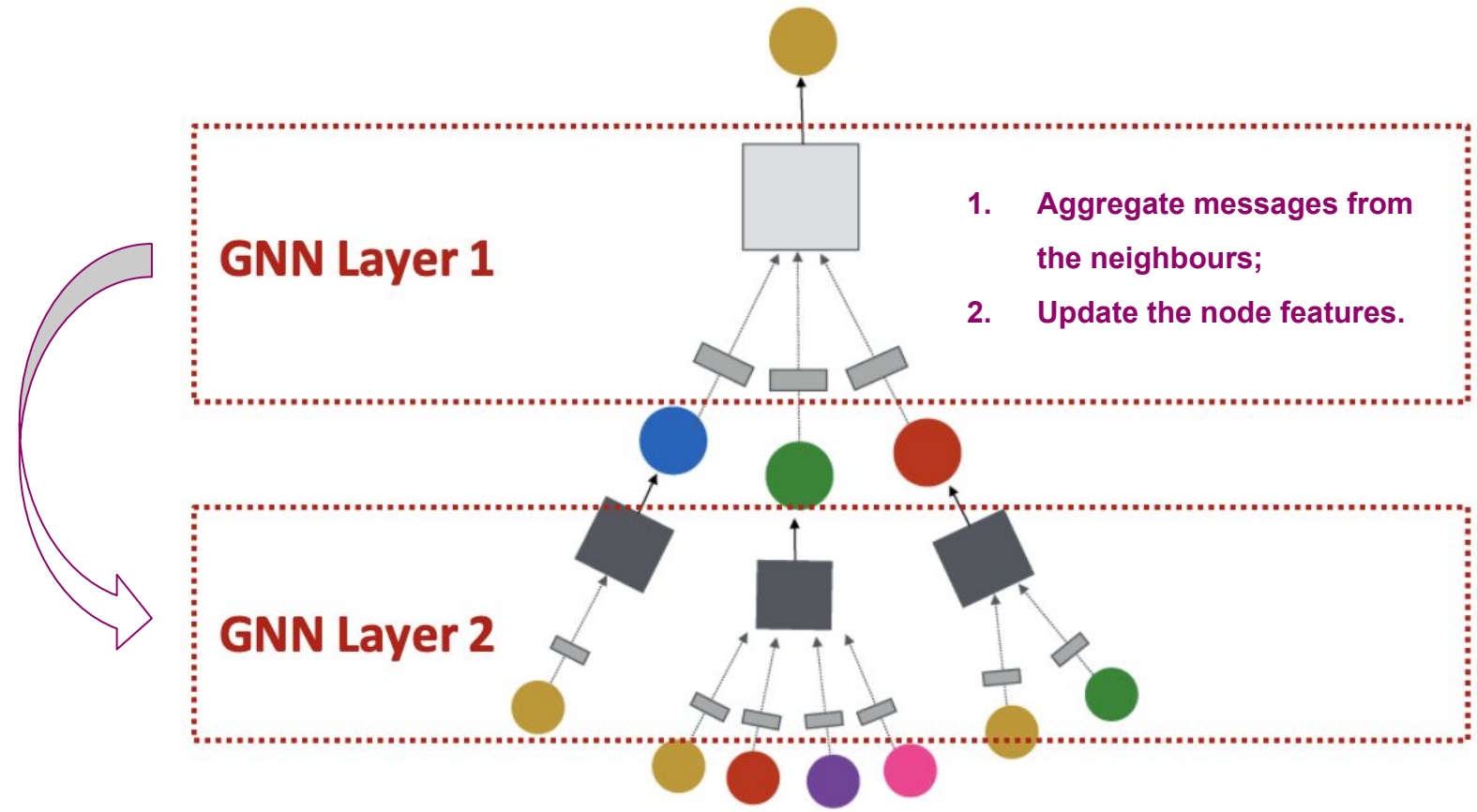
Credit: [1, 2]

Stacking GNN Layers



INPUT GRAPH

Layer connectivity



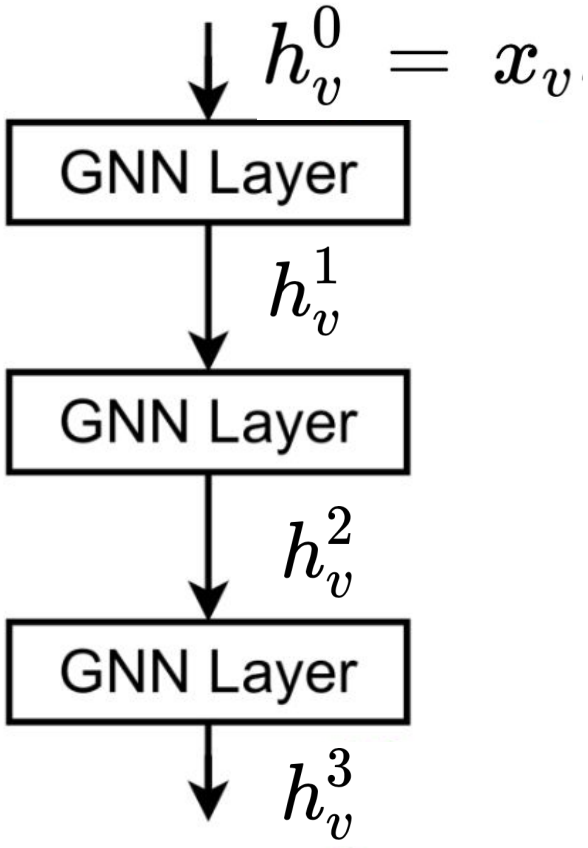
Credit: [1]

Stacking GNN Layers

The standard way: Stack GNN layers sequentially

Input: Initial raw node feature x_v

Output: Node embeddings after K GNN layers h_v^K

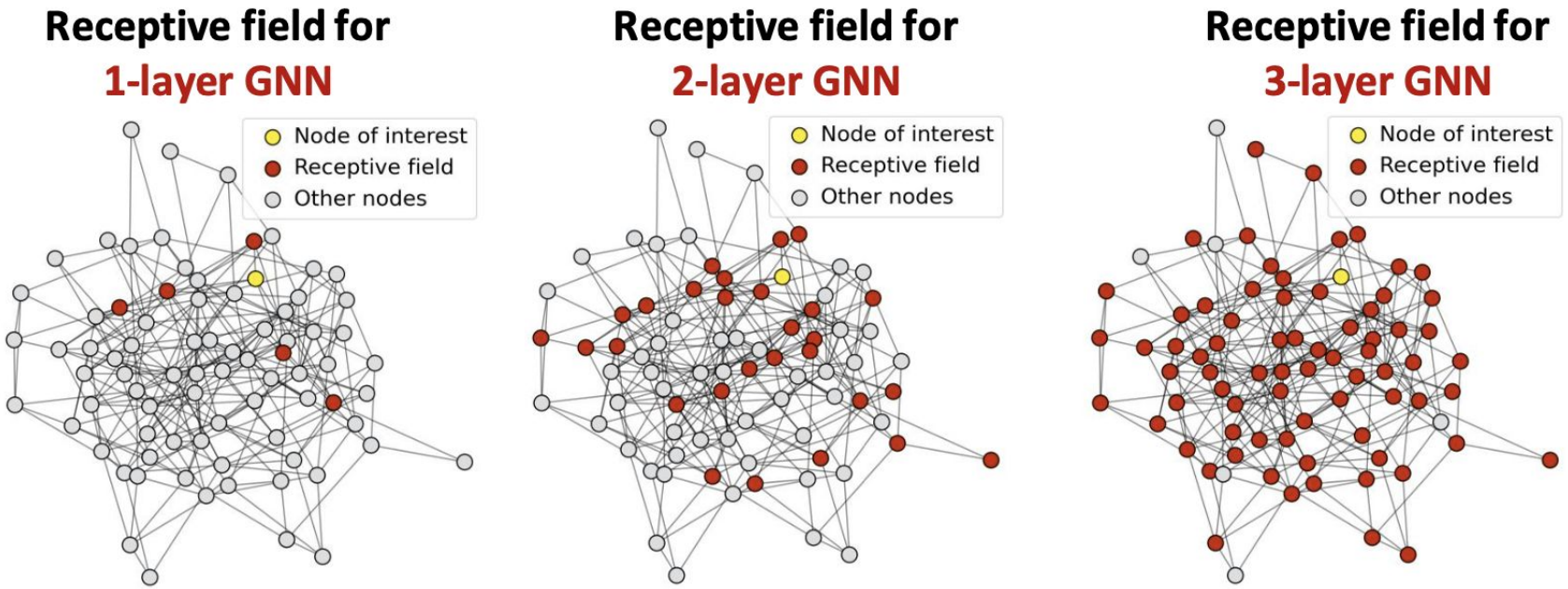


Over-Smoothing Problem

Over-Smoothing Problem

Over-smoothing: the issue of stacking many GNN layers → all the node embeddings converge to the same value.

Receptive field of GNN: the set of nodes that determine the embedding of a node of interest.

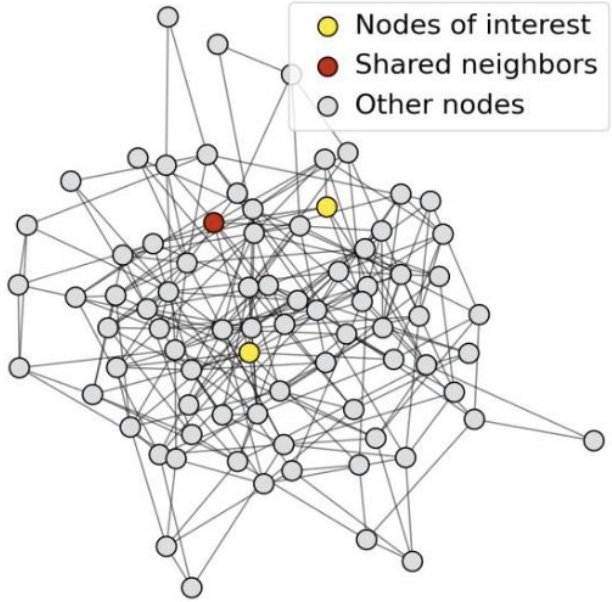


K-layer GNN: each node has a receptive field of K-hop neighborhood.

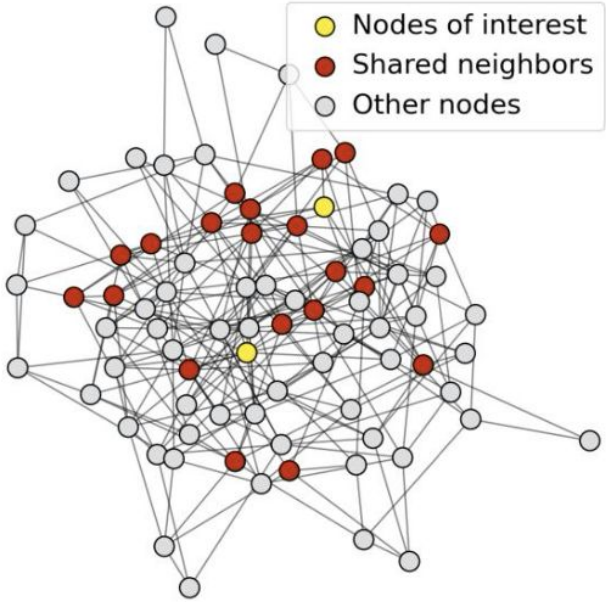
Over-Smoothing Problem

Receptive field overlap for two nodes: the shared neighbors quickly grows when we increase the number of hops (number of GNN layers).

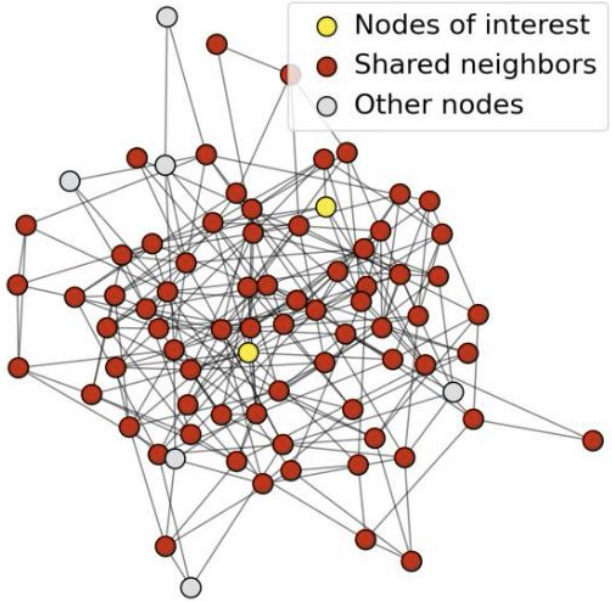
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes



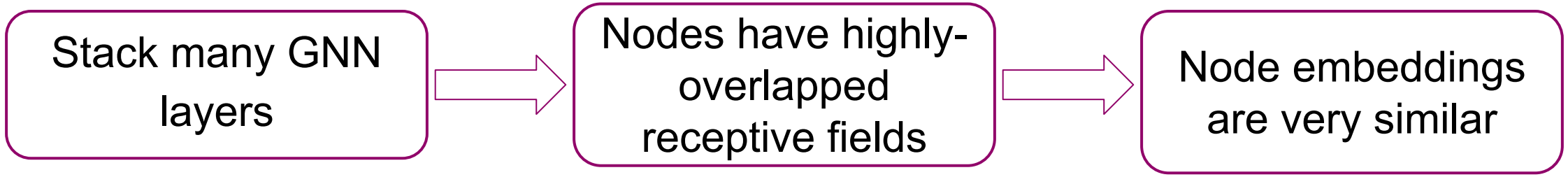
3-hop neighbor overlap
Almost all the nodes!



Credit: [1]

Over-Smoothing Problem & Receptive Field

Node embedding is determined by its receptive field → if two nodes have highly-overlapped receptive fields, then their embeddings are highly similar.



Why it is bad?

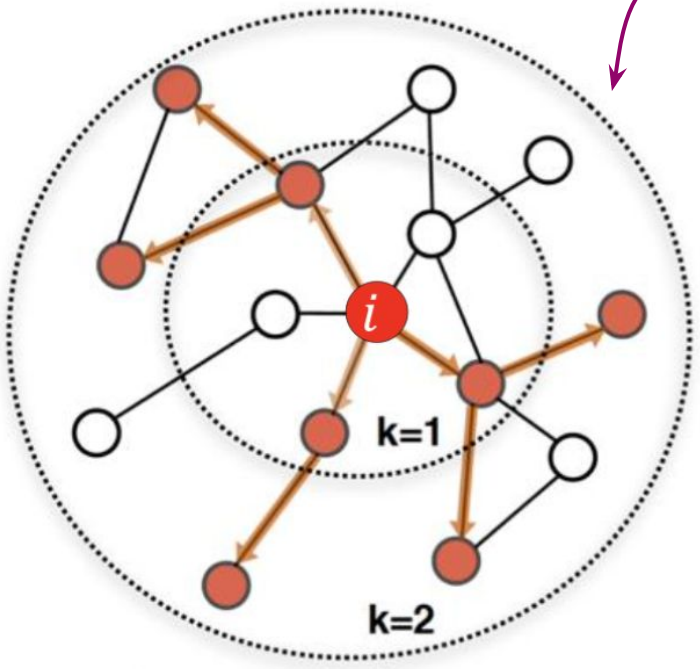
- Using node embeddings we are unable to differentiate the nodes.
- Node-specific information becomes “washed out” after several iterations of GNN message passing.

Credit: [1, 2]

Over-Smoothing Problem: General Advice

Be cautious when adding GNN layers!

Adding more GNN layers does not always help



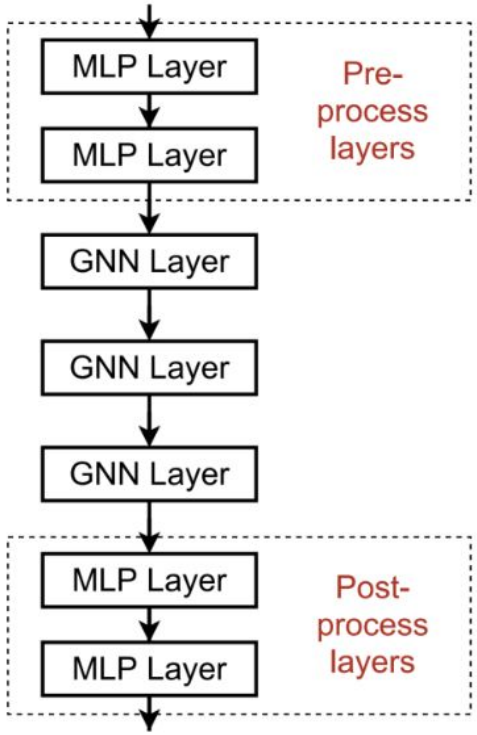
1. Analyze the necessary receptive field to solve your problem.
2. In Random Search set max number of GNN layers to be a bit more than the receptive field. Do not set it to be unnecessarily large!

Credit: [1, 2]

How to Improve: Expressive Power

Increase the expressive power within each GNN layer.

Add MLP layers (applied to each node) before and after GNN layers, as pre-process layers and post-process layers.

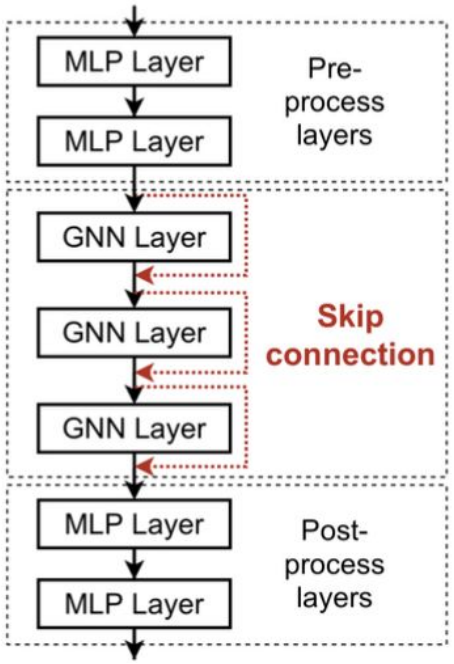


- Pre-processing layers: important when encoding node features is necessary.
- Post-processing layers: important when reasoning / transformation over node embeddings are needed.

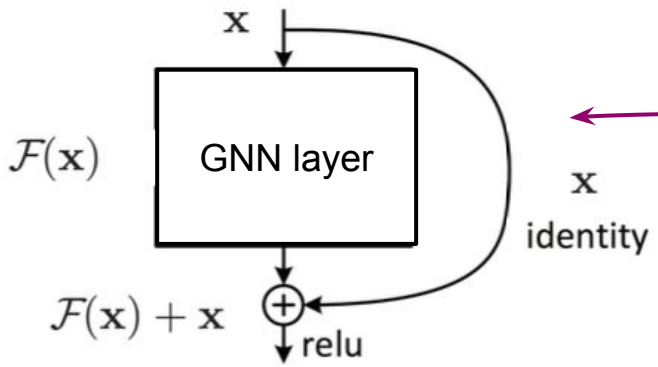
Credit: [1]

Design GNN Layer Connectivity: Adding Residual Connections

Add residual connections in GNN to prevent node attributes from becoming "washed out".



We can increase the impact of earlier layers on the final node embeddings.



Credit: [1, 4]

Design GNN Layer Connectivity: Adding Residual Connections

Recap:

$$h_v^k = f^k\left(W^k \underbrace{\frac{\sum_{u \in \mathcal{N}(v)} h_u^{k-1}}{|\mathcal{N}(v)|}}_{\text{AGGREGATE}} + B^k h_v^{k-1}\right), \forall v \in V$$

UPDATE

FINAL residual UPDATE = UPDATE + h_v^{k-1}

Sometimes these are also called skip connections.

Design GNN Layer Connectivity: Interpolation Method

Recap:

$$h_v^k = f^k\left(W^k \underbrace{\frac{\sum_{u \in \mathcal{N}(v)} h_u^{k-1}}{|\mathcal{N}(v)|}}_{\text{AGGREGATE}} + B^k h_v^{k-1}\right), \forall v \in V$$

UPDATE

FINAL UPDATE = α_1 UPDATE + $\alpha_2 h_v^{k-1}$

$\alpha_1, \alpha_2 \in [0,1]^d$ and $\alpha_2 = 1 - \alpha_1$

Design GNN Layer Connectivity: Alternative ways for Skip Connections

Recap:

$$h_v^k = f^k\left(W^k \frac{\sum_{u \in \mathcal{N}(v)} h_u^{k-1}}{|\mathcal{N}(v)|} + B^k h_v^{k-1}\right), \forall v \in V$$

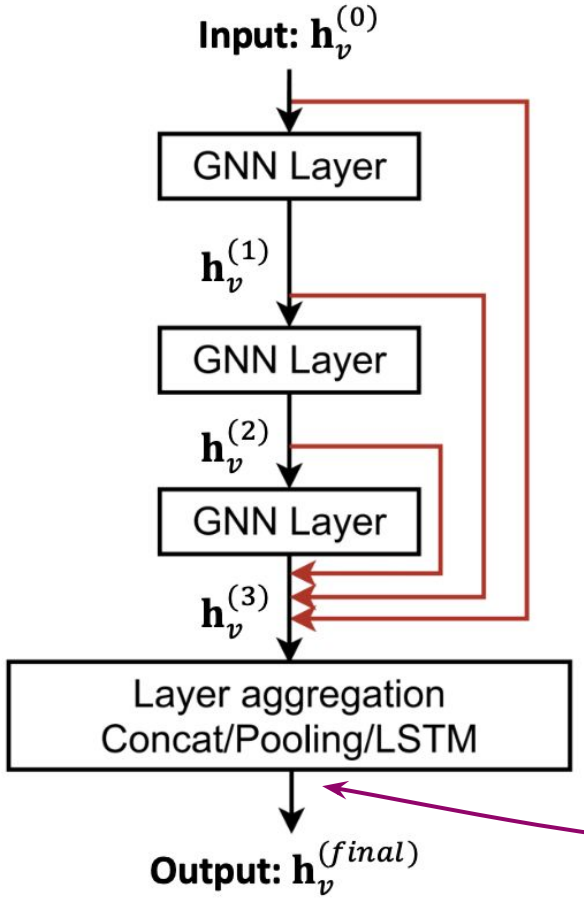
AGGREGATE

UPDATE

FINAL UPDATE = [UPDATE, h_v^{k-1}]

Key intuition: to encourage the model to disentangle information during message passing – separate the information coming from the neighbors from the current representation of each node.

Design GNN Layer Connectivity: Jumping Knowledge Connections



Leverage the representations at each layer of message passing, rather than only the output of the final layer.

$$h_v^{\text{final}} = f_{JK}\left[\underbrace{h_v^0, h_v^1, \ldots, h_v^K}_{\text{Concatenation}}\right]$$

Arbitrary differentiable function

Credit: [1, 2, 6]

Training Pipeline Construction

GNN Layers in Practice

Many DL modules can be incorporated into/with a GNN layer:

- Normalization – for speed training, preventing overtraining
  - Node level – batch, layer, etc → GraphNorm [12] (extension for the GNN).
- Dropout – for regularization, robustness
  - Node level - increasing the expressive power of GNN [10]
  - Edge level - increasing the expressive power of GNN, helps with over-fitting and over-smoothing [11]
  - Linear transformation

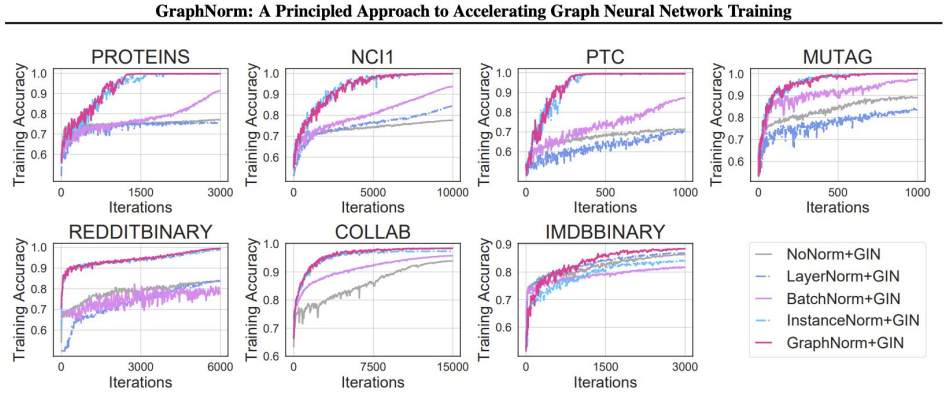


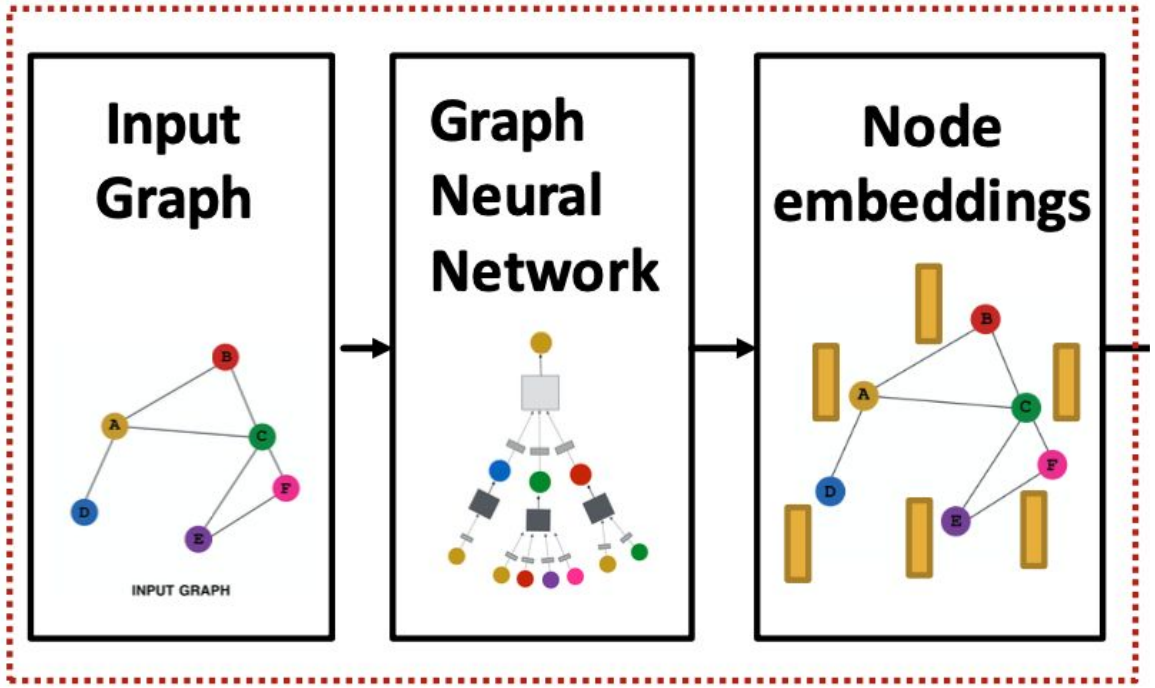
Figure 2. **Training performance** of GIN with different normalization methods and GIN without normalization in graph classification tasks. The convergence speed of our adaptation of InstanceNorm dominates BatchNorm and LayerNorm in most tasks. GraphNorm further improves the training over InstanceNorm especially on tasks with highly regular graphs, e.g., IMDB-BINARY (See Figure 5 for detailed illustration). Overall, GraphNorm converges faster than all other methods.

Depending on the task, combinations of different GNN layer variants

- E.g., network with GCN layers (convolution) followed by GAT layers (attention)

GNN Training Pipeline

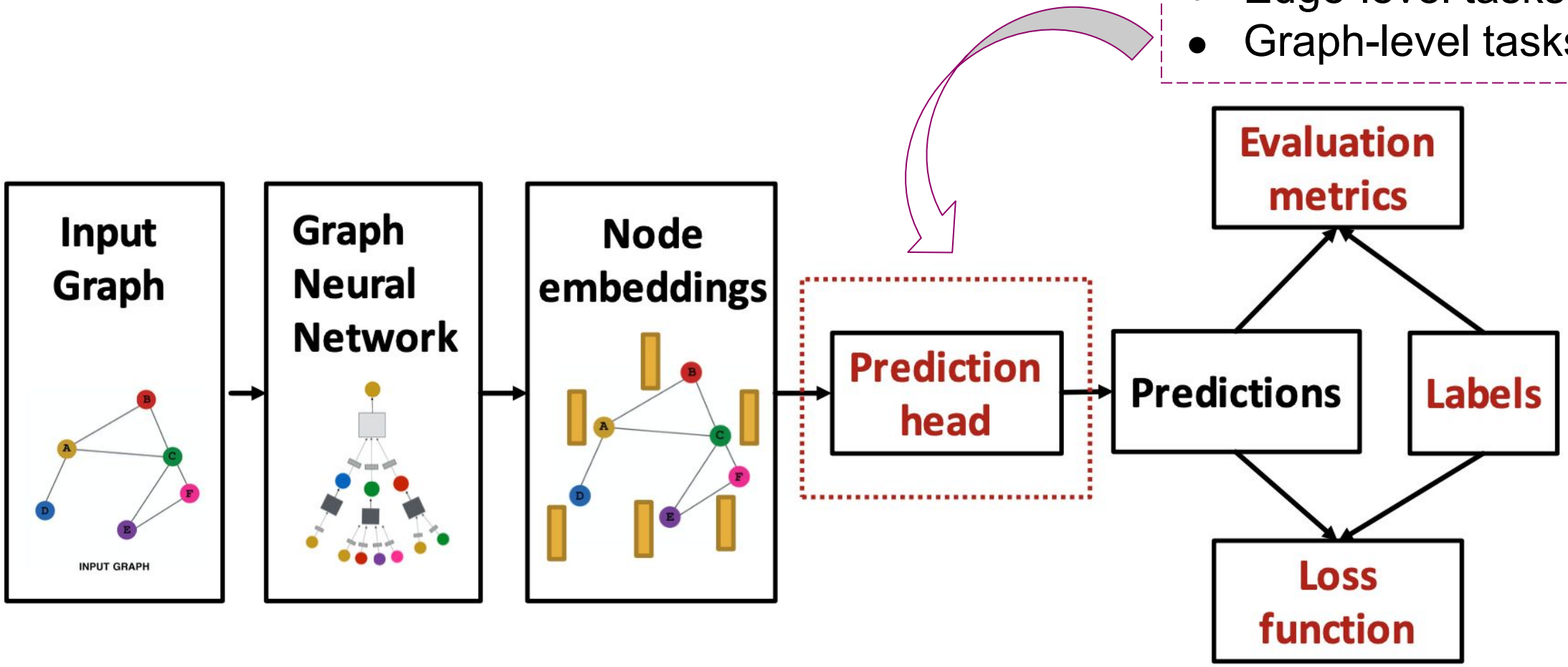
We covered this



Output of a GNN: set of node embeddings $h_v^K, \forall v \in V$

GNN Training Pipeline

- Node-level tasks
- Edge-level tasks
- Graph-level tasks



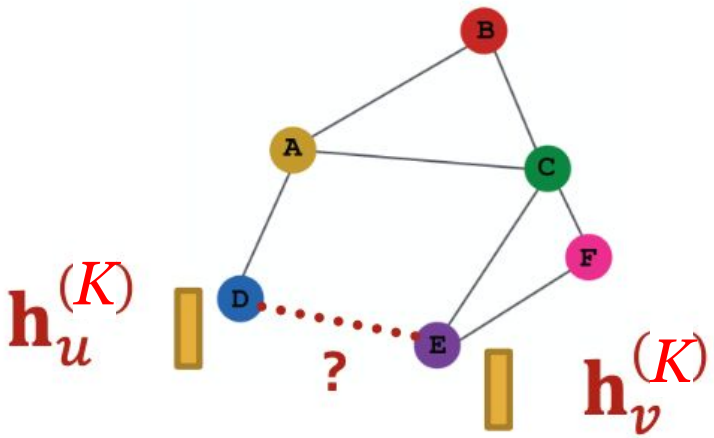
Credit: [1]

Prediction: Node & Edge Level

Node-level prediction: directly make prediction using node embeddings $h_v^K, \forall v \in V$

- Classification: classify among c categories.
- Regression: regress on c targets.

Link-level prediction: predict the link between two nodes, make prediction using pairs of node embeddings.

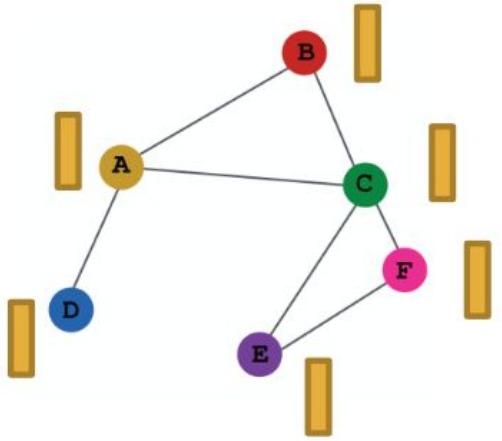


1. Linear(Concatenate $\left[h_u^K, h_v^K\right]$),
2. Average,
3. Hadamard $h_u^K * h_v^K$,
4. Dot Product,
5. Weighted L-1, L-2.

Credit: [1, 7]

Prediction: Graph Level

Graph-level prediction: make prediction using all the node embeddings in our graph.



Graph-level prediction

$$\hat{y}_G = \text{PredictionHead}\left(\left\{h_v^K, \forall v \in V\right\}\right)$$

1. Global mean pooling $\hat{y}_G = \text{Mean}\left(\left\{h_v^K, \forall v \in V\right\}\right)$
2. Global max pooling $\hat{y}_G = \text{Max}\left(\left\{h_v^K, \forall v \in V\right\}\right)$
3. Global sum pooling $\hat{y}_G = \text{Sum}\left(\left\{h_v^K, \forall v \in V\right\}\right)$

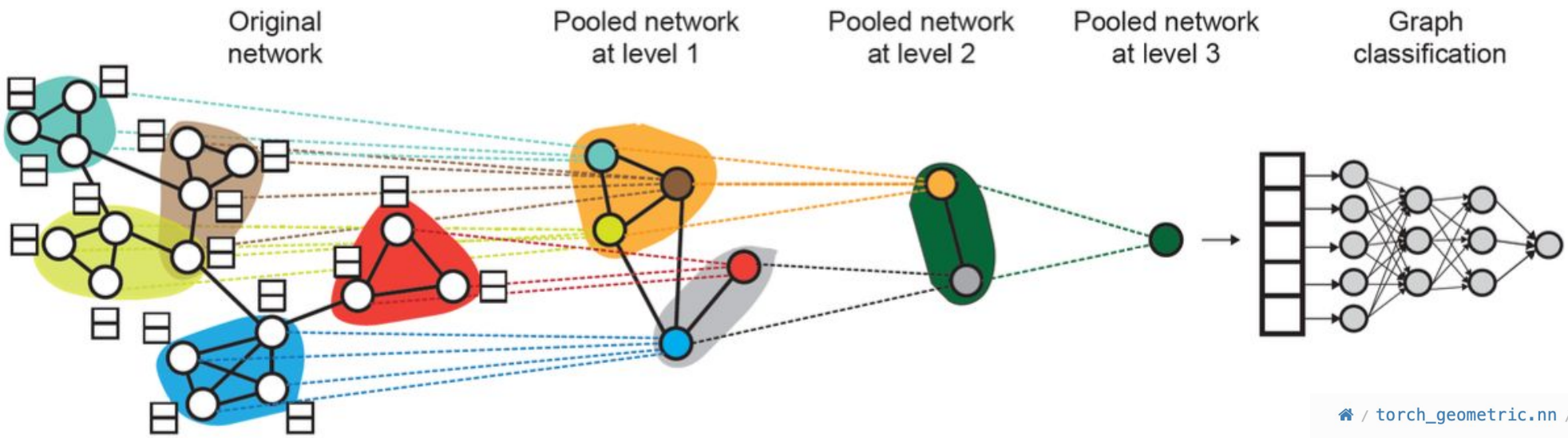
Issue: Global pooling over a (large) graph will lose information.

Credit: [1]

Graph Pooling

1. Attention-based Pooling [8].

2. Hierarchical Pooling - graph clustering to perform the pooling [9].



/ torch_geometric.nn / dense.dense_diff_pool

dense.dense_diff_pool

Credit: [8, 9]

Other Pooling Methods

Pooling Layers

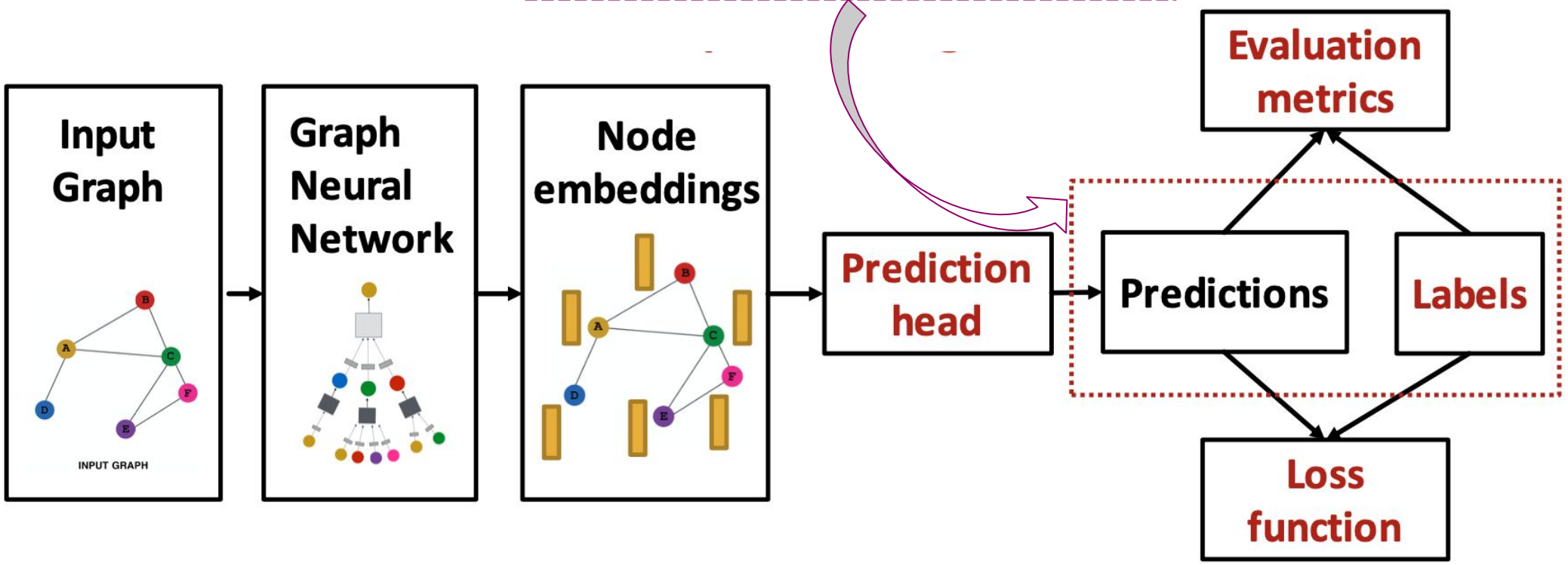
`global_add_pool`	Returns batch-wise graph-level-outputs by adding node features across the node dimension, so that for a single graph $\mathcal{G}_i$ its output is computed by
`global_mean_pool`	Returns batch-wise graph-level-outputs by averaging node features across the node dimension, so that for a single graph $\mathcal{G}_i$ its output is computed by
`global_max_pool`	Returns batch-wise graph-level-outputs by taking the channel-wise maximum across the node dimension, so that for a single graph $\mathcal{G}_i$ its output is computed by
`KNNIndex`	A base class to perform fast k-nearest neighbor search (k-NN) via the `faiss` library.
`L2KNNIndex`	Performs fast k-nearest neighbor search (k-NN) based on the L_2 metric via the `faiss` library.
`MIPSKNNIndex`	Performs fast k-nearest neighbor search (k-NN) based on the maximum inner product via the `faiss` library.
`TopKPooling`	top_k pooling operator from the "Graph U-Nets", "Towards Sparse Hierarchical Graph Classifiers" and "Understanding Attention and Generalization in Graph Neural Networks" papers.
`SAGPooling`	The self-attention pooling operator from the "Self-Attention Graph Pooling" and "Understanding Attention and Generalization in Graph Neural Networks" papers.
`EdgePooling`	The edge pooling operator from the "Towards Graph Pooling by Edge Contraction" and "Edge Contraction Pooling for Graph Neural Networks" papers.
`ASAPooling`	The Adaptive Structure Aware Pooling operator from the "ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations" paper.
`PANPooling`	The path integral based pooling operator from the "Path Integral Based Convolution and Pooling for Graph Neural Networks" paper.
`MemPooling`	Memory based pooling layer from "Memory-Based Graph Networks" paper, which learns a coarsened graph representation based on soft cluster assignments
`max_pool`	Pools and coarsens a graph given by the `torch_geometric.data.Data` object according to the clustering defined in `cluster`.
`avg_pool`	Pools and coarsens a graph given by the `torch_geometric.data.Data` object according to the clustering defined in `cluster`.

Docs for PyTorch Geometric: Pooling Layers

GNN Training Pipeline

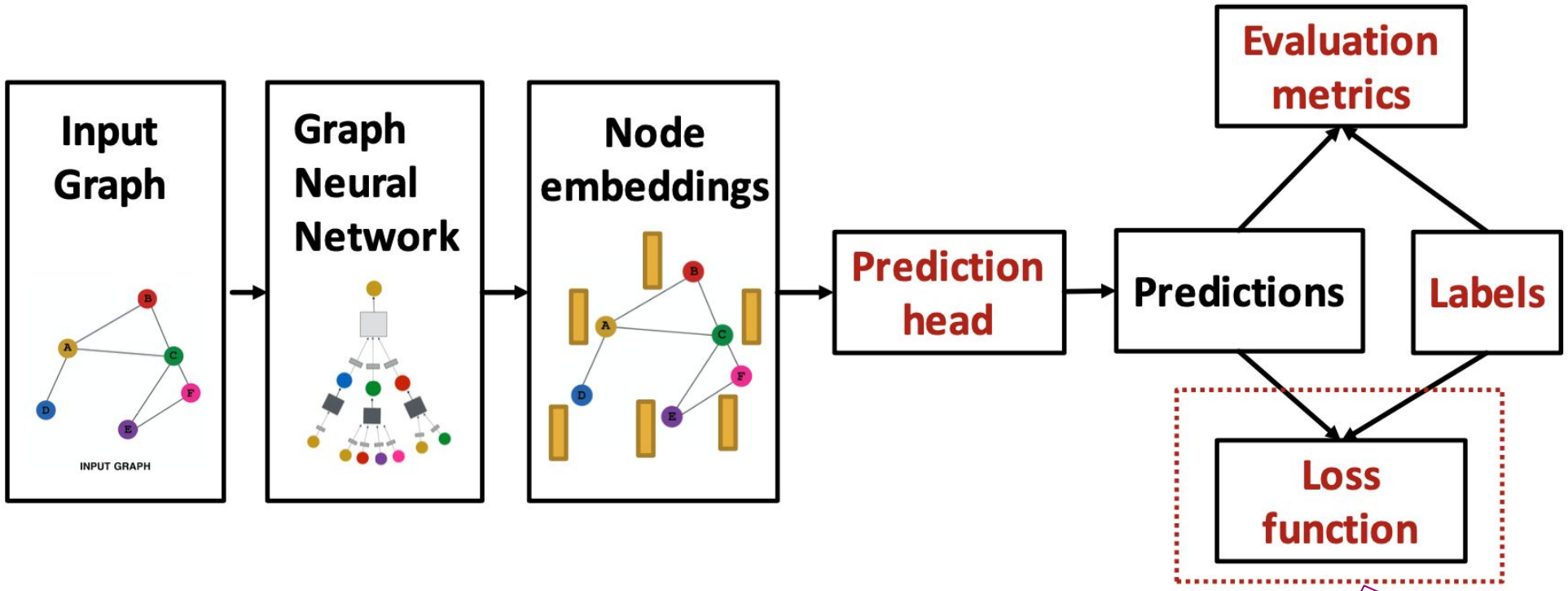
Depends on the task/settings:

- Supervised labels,
- Unsupervised signals.



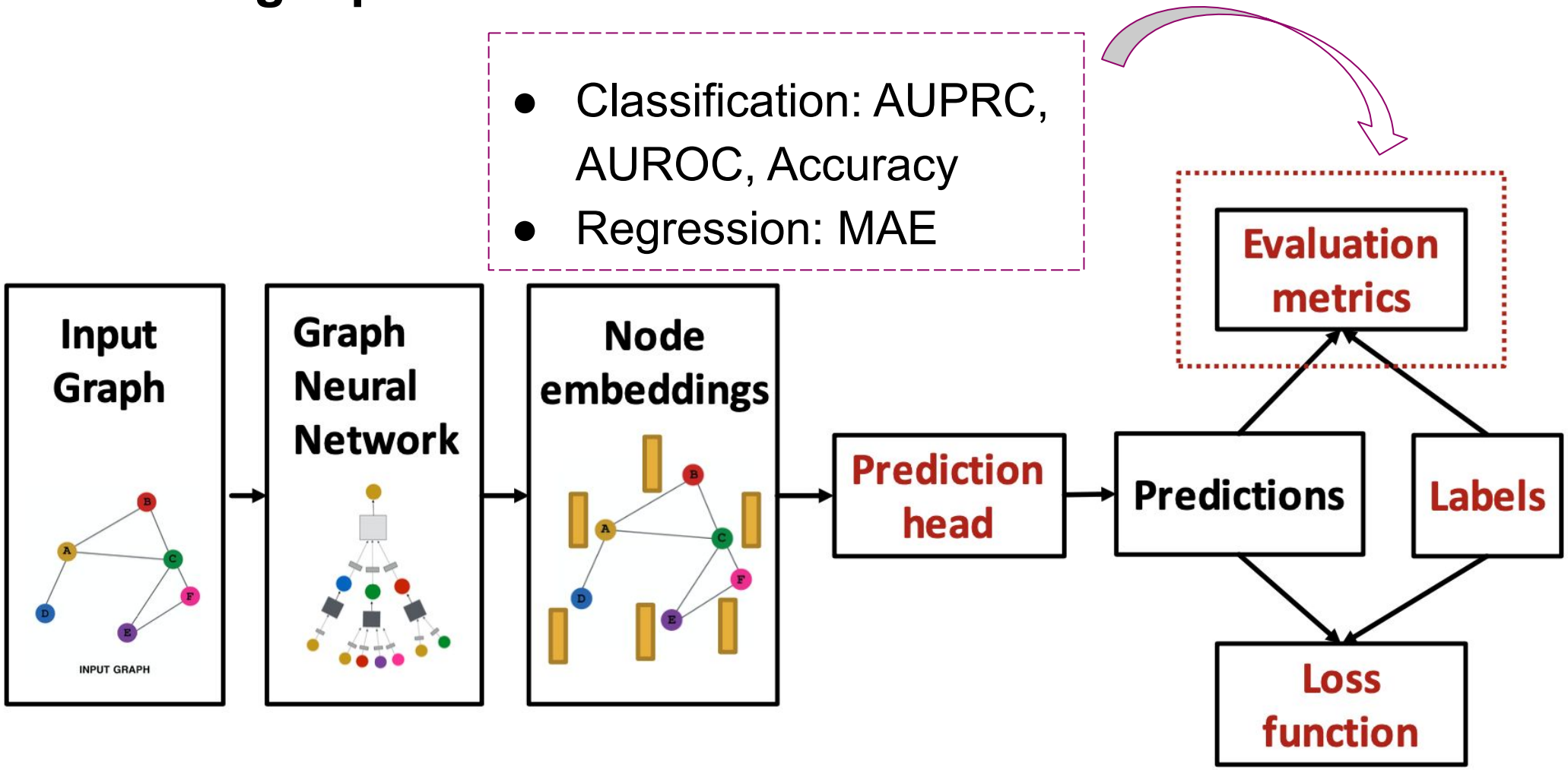
Credit: [1]

GNN Training Pipeline

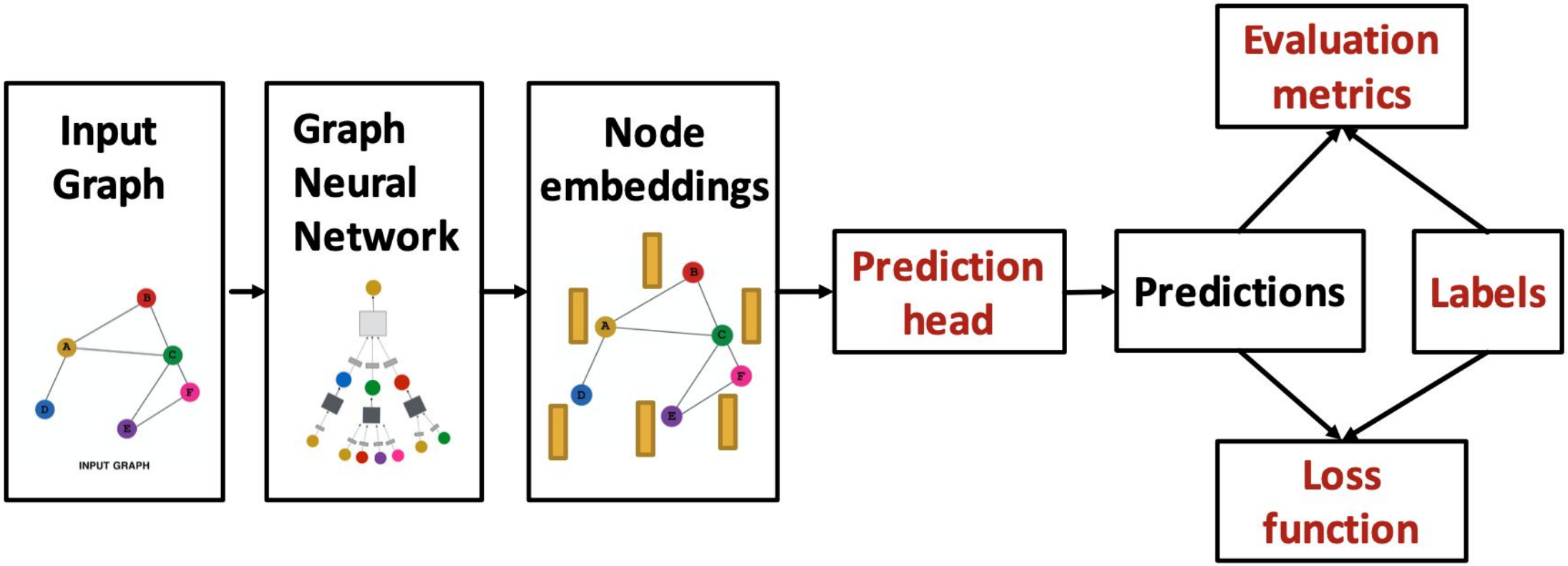


- Classification: Cross-entropy
- Regression: MSE

GNN Training Pipeline



GNN Training Pipeline



How do we split our dataset into train / validation / test set?

Credit: [1]

Why Splitting Graphs is special

Often, the data points are independent (i.e image classification).

Splitting a graph dataset is different!

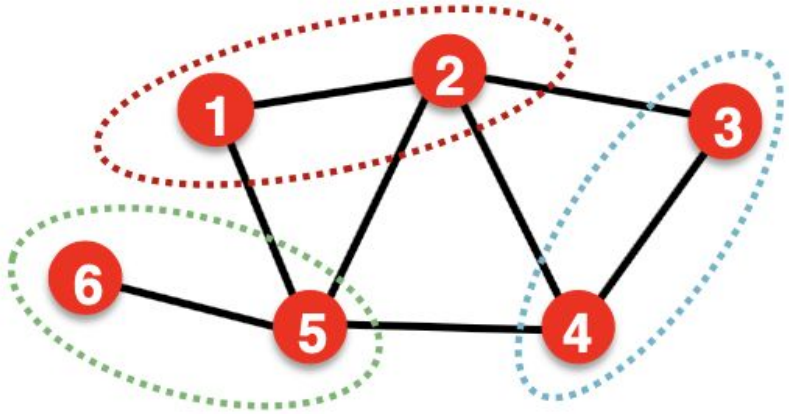
Example:

Node classification: data points are NOT independent.

Training

Validation

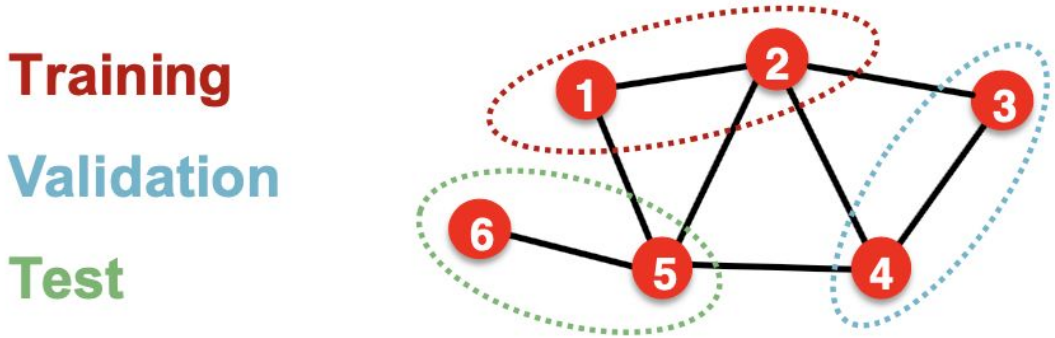
Test



Node 5 will affect our prediction on node 1, because it will participate in message passing→affect node 1's embedding.

Transductive setting

The input graph can be observed in all the dataset splits (training /validation /test set). We only split the labels.



1. At training time: we compute embeddings using the entire graph, and train using node 1&2's labels → i.e these labels are the part of the supervised loss computation.
2. At validation time: we compute embeddings using the entire graph, and evaluate on node 3&4's labels.

Transductive setting

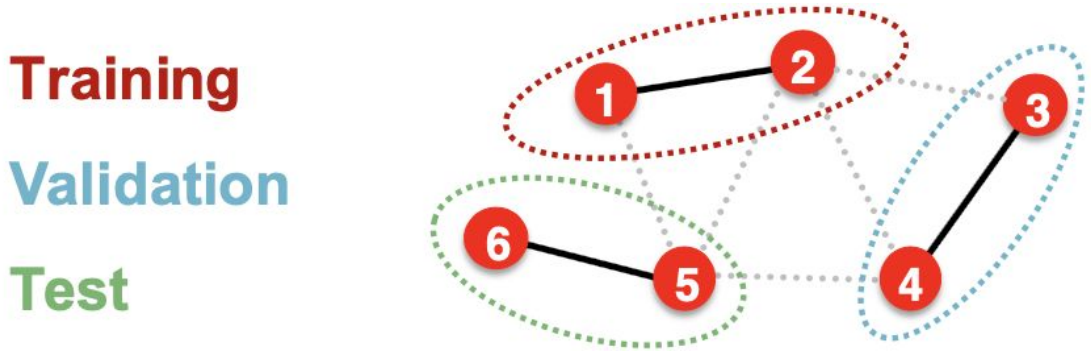
The input graph can be observed in all the dataset splits (training /validation /test set). We only split the labels.

- The dataset consists of one graph.
- The entire graph can be observed in all dataset splits, we only split the labels.
- Only applicable to node / edge prediction tasks.

Use case: graph is static and won't change in the future.

Inductive setting

We break the edges to create the multiple graphs. Training / validation / test sets are on different graphs.



1. At training time: we compute embeddings using the graph over node 1&2, and train using node 1&2's labels.
2. At validation time: we compute embeddings using the graph over node 3&4, and evaluate on node 3&4's labels.

Credit: [1,2]

Inductive setting

We break the edges to create the multiple graphs. Training / validation / test sets are on different graphs.

- The dataset consists of multiple graphs.
- Each split can only observe the graph(s) within the split. A successful model should generalize to unseen graphs.
- Applicable to node / edge / graph tasks.

Use case: graph is dynamic.

Take-Home Messages

1. Stacking GNN layers: what to do with over-smoothing
    a. Deciding number of layers carefully;
    b. Residual and skip connections,
2. The full training pipeline of a GNN:
    a. The output of a GNN: node embeddings,
    b. Different prediction heads: node/edge/graph level,
    c. Losses & Metrics,
    d. How to split the data: Transductive and Inductive settings.

BIOMEDICAL INFORMATICS

Slides & Image Credits

1. CS224W: Machine Learning with Graphs
2. Graph Representation Learning Book
3. Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges
4. Deep Residual Learning for Image Recognition
5. Column Networks for Collective Classification
6. Representation Learning on Graphs with Jumping Knowledge Networks
7. node2vec: Scalable Feature Learning for Networks
8. ORDER MATTERS: SEQUENCE TO SEQUENCE FOR SETS
9. Hierarchical Graph Representation Learning with Differentiable Pooling
10. DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks
11. DROPEDGE: TOWARDS DEEP GRAPH CONVOLU- TIONAL NETWORKS ON NODE CLASSIFICATION
12. GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training