GNN Augmentation and Training

https://web.stanford.edu/class/cs224w/slides/04-GNN2.pdf
https://web.stanford.edu/class/cs224w/slides/05-GNN3.pdf
https://web.stanford.edu/class/cs224w/slides/06-theory.pdf

GraphCL:
https://docs.google.com/presentation/d/1ZSyXdvDuYcJdfGYXUqyId8ZWCd_1hPtM/edit?usp=sharing&ouid=114807689254736809193&rtpof=true&sd=true
https://proceedings.neurips.cc/paper_files/paper/2020/hash/3fe230348e9a12c13120749e3f9fa4cd-Abstract.html

Caltech EAS AI BOOTCAMP

Over-smoothing

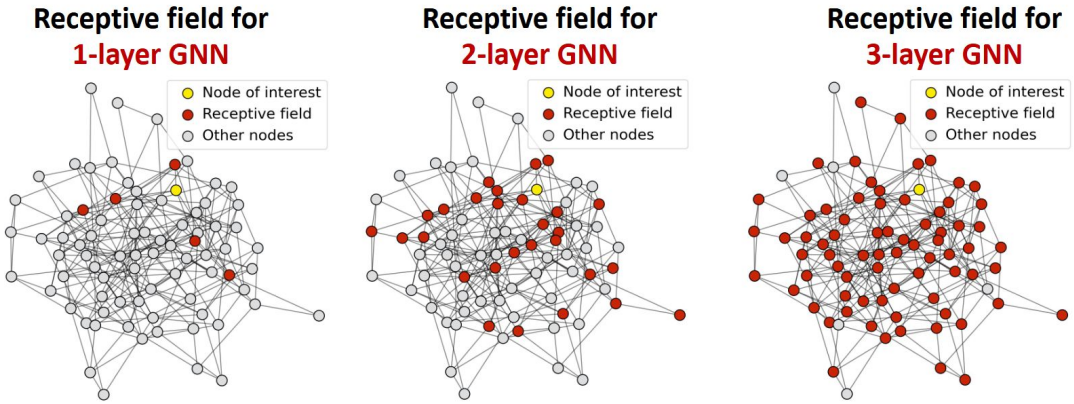
Over-smoothing

- **The issue of stacking many GNN layers**
  - GNN suffers from **the over-smoothing problem**
- **The over-smoothing problem: all the node embeddings converge to the same value**
  - This is bad because we **want to use node embeddings to differentiate nodes**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Receptive Field of a GNN

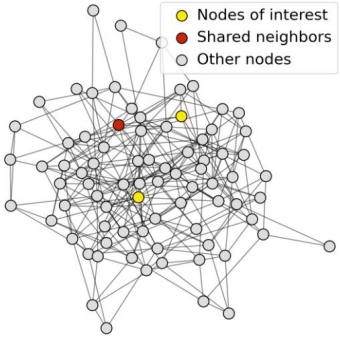
- **Receptive field:** the set of nodes that determine the embedding of a node of interest
  - **In a K-layer GNN, each node has a receptive field of K-hop neighborhood**



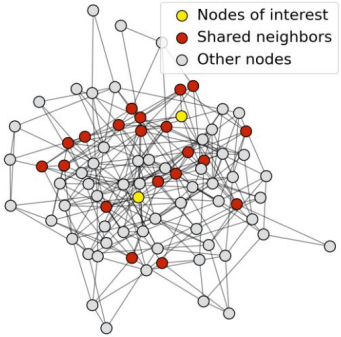
Receptive Field of a GNN

- **Receptive field overlap** for two nodes
  - **The shared neighbors quickly grows** when we increase the number of hops (num of GNN layers)

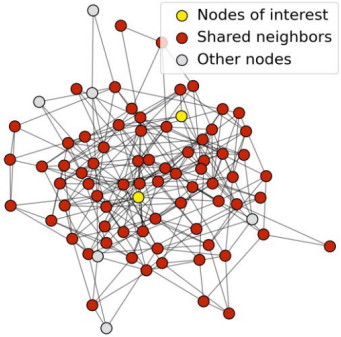
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes



3-hop neighbor overlap
Almost all the nodes!



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Receptive Field and Over-smoothing

- **We can explain over-smoothing via the notion of the receptive field**
  - We know **the embedding of a node is determined by its receptive field**
    - If two nodes **have highly-overlapped receptive fields, then their embeddings are highly similar**
  - **Stack many GNN layers → nodes will have highly-overlapped receptive fields → node embeddings will be highly similar → suffer from the over-smoothing problem**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Strategy 1: Number of GNN Layers

- **Be cautious when adding GNN layers**
  - Unlike neural networks in other domains (CNN for image classification), **adding more GNN layers do not always help**
  - **Step 1: Analyze the necessary receptive field** to solve your problem. E.g., by computing the diameter of the graph
  - **Step 2:** Set number of GNN layers L to be a bit more than the receptive field we like. **Do not set L to be unnecessarily large**!

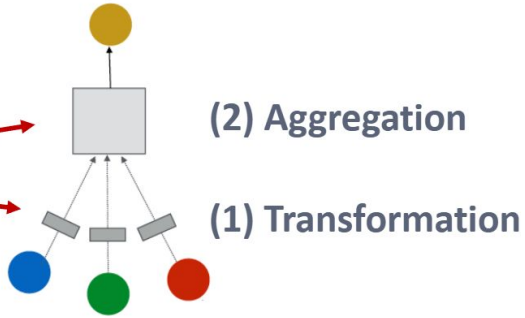
- **Question:** How to enhance the expressive power of a GNN, if the number of GNN layers is small?

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Improve Expressiveness of a Shallow GNN

- **Solution 1:** Increase the expressive power **within each GNN layer**
  - In our previous examples, each transformation or aggregation function only include one linear layer
  - We can **make aggregation / transformation become a deep neural network**!

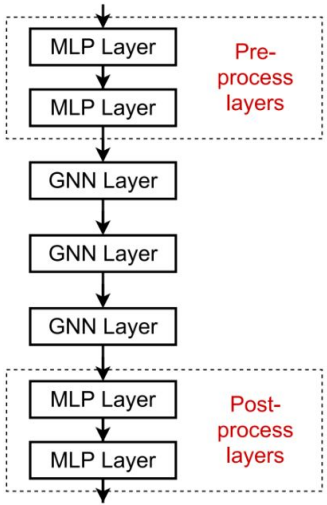
If needed, each box could include a 3-layer MLP



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Improve Expressiveness of a Shallow GNN

- **Solution 2:** Add layers that do not pass messages
  - A GNN does not necessarily only contain GNN layers
    - E.g., we can 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.
E.g., when nodes represent images/text

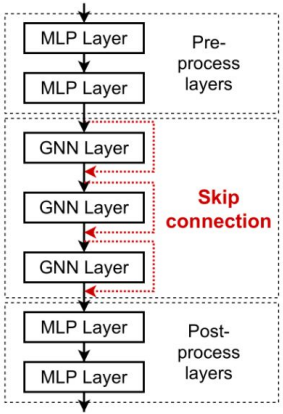
Post-processing layers: Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

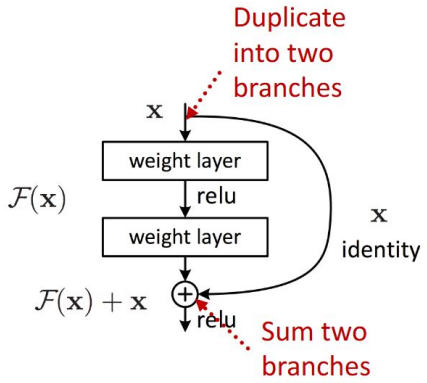
In practice, adding these layers works great!

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Strategy 2: Skip Connections

- **What if my problem still requires many GNN layers?**
- **Add skip connections in GNNs**
  - **Observation from over-smoothing:** Node embeddings in earlier GNN layers can sometimes better differentiate nodes
  - **Solution:** We can increase the impact of earlier layers on the final node embeddings, **by adding shortcuts in GNN**





Idea of skip connections:

Before adding shortcuts:

$$\boldsymbol{F}(\mathbf{x})$$

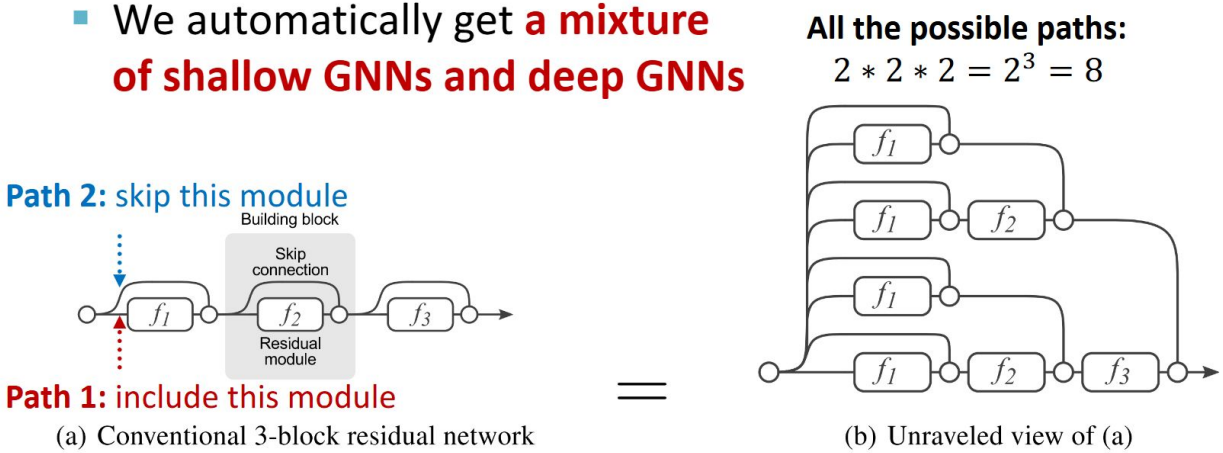
After adding shortcuts:

$$\boldsymbol{F}(\mathbf{x}) + \mathbf{x}$$

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Idea of Skip Connections

- **Why do skip connections work?**
  - **Intuition:** Skip connections create **a mixture of models**
  - N skip connections → 2^N possible paths
  - Each path could have up to N modules
  - We automatically get **a mixture of shallow GNNs and deep GNNs**



(a) Conventional 3-block residual network

(b) Unraveled view of (a)

Veit et al. Residual Networks Behave Like Ensembles of Relatively Shallow Networks, ArXiv 2016

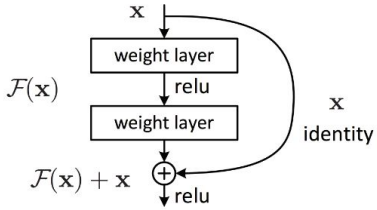
Taken from Stanford CS224W course: http://cs244w.stanford.edu

GCN with Skip Connections

- **A standard GCN layer**

- $\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$

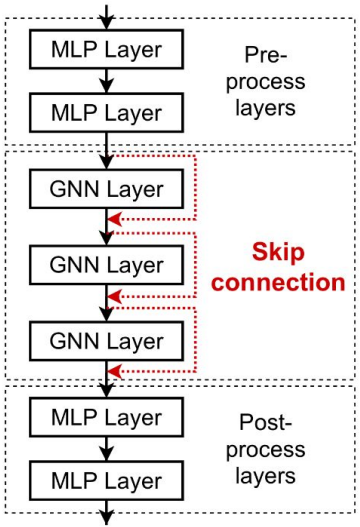
This is our $F(\mathbf{x})$



- **A GCN layer with skip connection**

- $\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} + \mathbf{h}_v^{(l-1)}\right)$

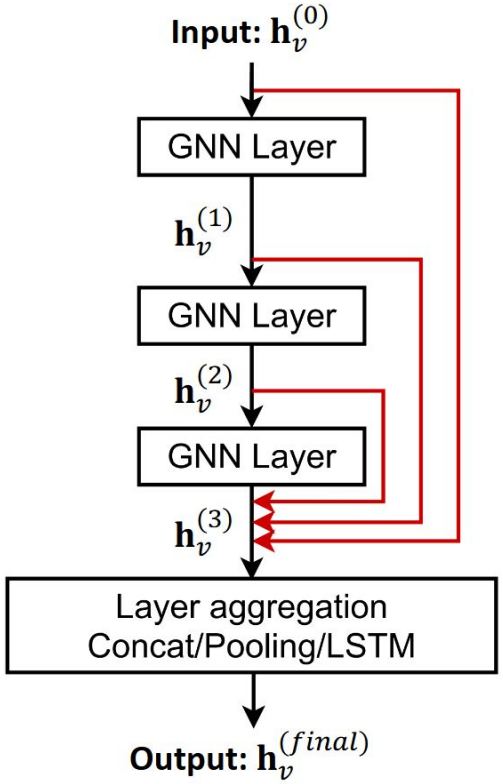
$F(\mathbf{x})$ + $\mathbf{x}$



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Other Options of Skip Connections

- **Other options:** Directly skip to the last layer
  - The final layer directly **aggregates from the all the node embeddings** in the previous layers



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Graph Manipulation

Reasons for Manipulating Graphs

- **Feature level:**
  - The input graph **lacks features** → feature augmentation
- **Structure level:**
  - The graph is **too sparse** → inefficient message passing
  - The graph is **too dense** → message passing is too costly
  - The graph is **too large** → cannot fit the computational graph into a GPU
- It's just **unlikely that the input graph happens to be the optimal computation graph** for embeddings

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Graph Structure Manipulation

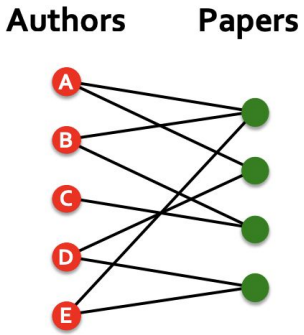
- The graph is **too sparse** → **Add virtual nodes / edges**
- The graph is **too dense** → **Sample neighbors when doing message passing**
- The graph is **too large** → **Sample subgraphs to compute embeddings**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Adding Virtual Edges

- **Common approach:** Connect 2-hop neighbors via virtual edges
- **Intuition:** Instead of using adj. matrix A for GNN computation, use $A + A^2$

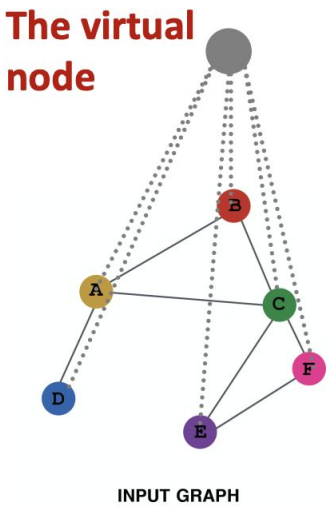
- **Use cases:** Bipartite graphs
  - Author-to-papers (they authored)
  - 2-hop virtual edges make an author-author collaboration graph



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Adding Virtual Nodes

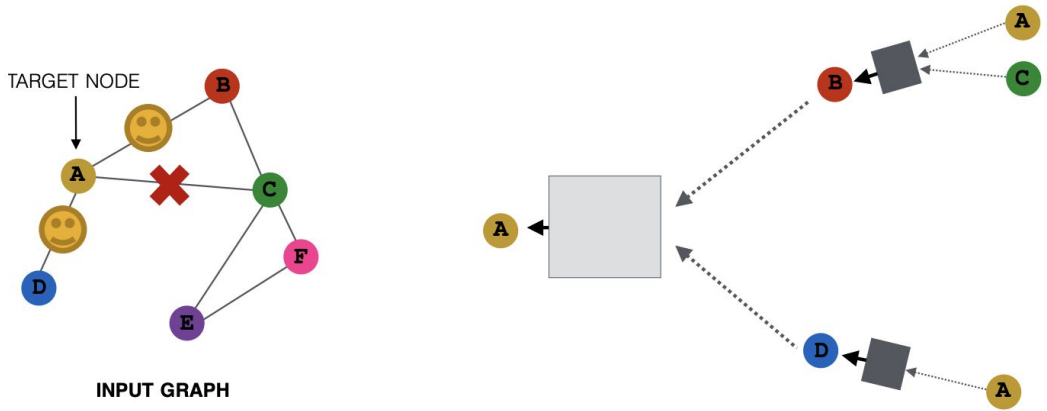
- The virtual node will connect to all the nodes in the graph
  - Suppose in a sparse graph, two nodes have shortest path distance of 10
  - After adding the virtual node, **all the nodes will have a distance of 2**
    - **Node A – Virtual node – Node B**
- **Benefits:** Greatly **improves message passing in sparse graphs**



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Random Edge Sampling

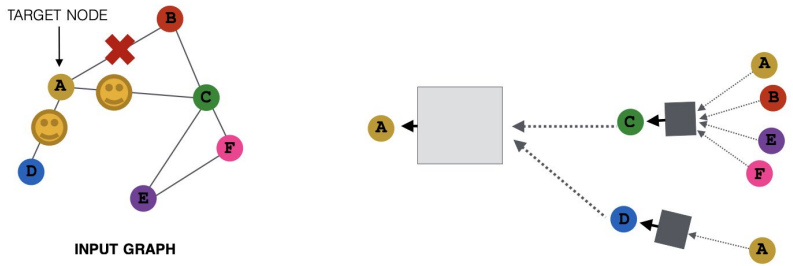
- Only nodes B and D will pass message to A



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Random Edge Sampling

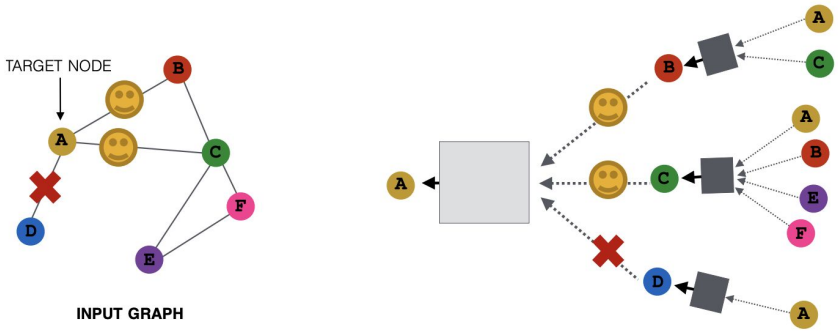
- **Next time when we compute the embeddings, we can sample different neighbors**
  - Only nodes C and D will pass message to A



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Random Edge Sampling

- In expectation, we can get embeddings similar to the case where all the neighbors are used
  - **Benefits:** Greatly reduce computational cost
  - And in practice it works great!

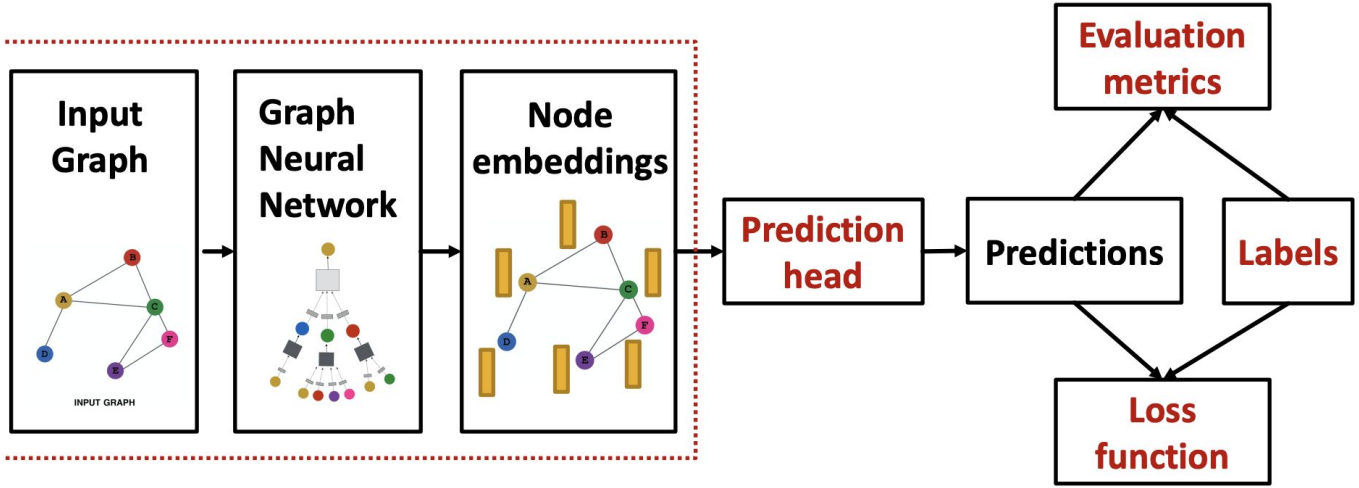


Taken from Stanford CS224W course: http://cs244w.stanford.edu

GNN Training Pipeline

Caltech EAS AI BOOTCAMP

Using Node Embedding in Prediction

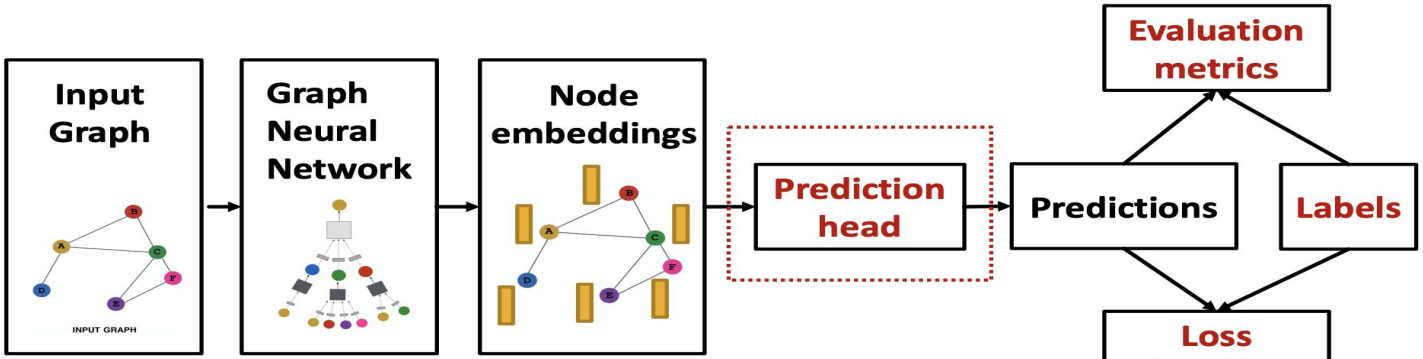


Output of a GNN: set of node embeddings

$$\{\mathbf{h}_v^{(L)}, \forall v \in G\}$$

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Prediction Head



(1) Different prediction heads:

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

Taken from Stanford CS224W course: http://cs244w.stanford.edu

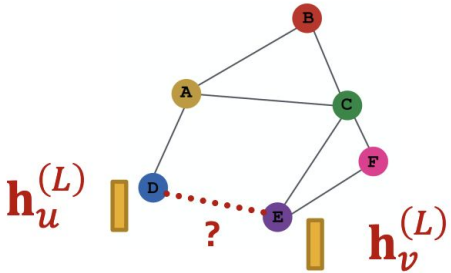
Node Level Prediction

- After GNN computation, we have d-dim node embeddings: $\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\}$
- Suppose we want to make k-way prediction
  - Classification: classify among k categories
  - Regression: regress on k targets
- $\hat{\boldsymbol{y}}_v = \text{Head}_{\text{node}}(\mathbf{h}_v^{(L)}) = \mathbf{W}^{(H)}\mathbf{h}_v^{(L)}$
  - $\mathbf{W}^{(H)} \in \mathbb{R}^{k \times d}$: We map node embeddings from $\mathbf{h}_v^{(L)} \in \mathbb{R}^d$ to $\hat{\boldsymbol{y}}_v \in \mathbb{R}^k$ so that we can compute the loss

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Edge Level Prediction

- Suppose we want to make k-way prediction
- $\hat{\boldsymbol{y}}_{\boldsymbol{uv}} = \text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$

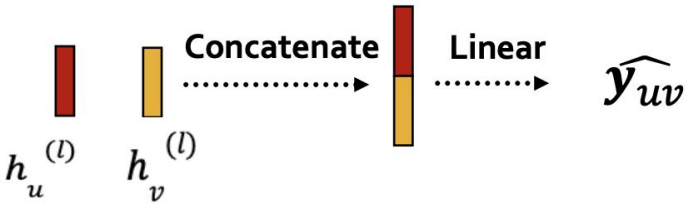


- What are the options for $\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$?

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Edge Level Prediction -Concat + Linear

- We have seen this in graph attention



- $\hat{\boldsymbol{y}}_{uv} = \text{Linear}(\text{Concat}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}))$
- Here $\text{Linear}(\cdot)$ will map $2d$-dimensional embeddings (since we concatenated embeddings) to k-dim embeddings (k-way prediction)

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Edge Level Prediction - Dot Product

- $\hat{y}_{uv} = (\mathbf{h}_u^{(L)})^T \mathbf{h}_v^{(L)}$
- **This approach only applies to 1-way prediction** (e.g., link prediction: predict the existence of an edge)
- **Applying to k-way prediction:**
  - Similar to **multi-head attention**: $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(k)}$ trainable

$$\hat{y}_{uv}^{(1)} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(1)} \mathbf{h}_v^{(L)}$$

$$\ldots$$

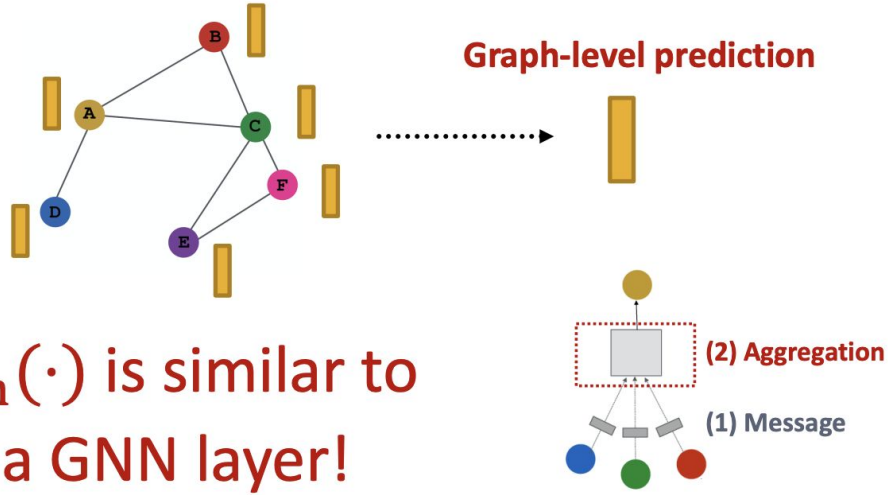
$$\hat{y}_{uv}^{(k)} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(k)} \mathbf{h}_v^{(L)}$$

$$\hat{y}_{uv} = \text{Concat}(\hat{y}_{uv}^{(1)}, \ldots, \hat{y}_{uv}^{(k)}) \in \mathbb{R}^k$$

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Graph Level Prediction

- Suppose we want to make k-way prediction
- $\hat{\boldsymbol{y}}_G = \text{Head}_{\text{graph}}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$



- $\text{Head}_{\text{graph}}(\cdot)$ is similar to $\text{AGG}(\cdot)$ in a GNN layer!

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Graph Level Prediction

- Options for $\text{Head}_{\text{graph}}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$
- **(1) Global mean pooling**

$$\hat{\boldsymbol{y}}_G = \text{Mean}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- **(2) Global max pooling**

$$\hat{\boldsymbol{y}}_G = \text{Max}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- **(3) Global sum pooling**

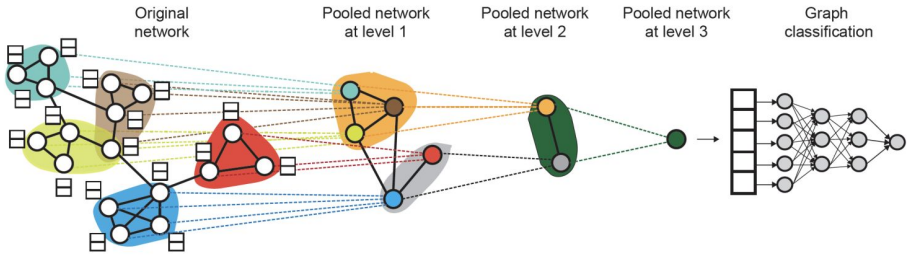
$$\hat{\boldsymbol{y}}_G = \text{Sum}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- These options work great for small graphs
- **Can we do better for large graphs?**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Hierarchical Graph Pooling – Diffpool

- **DiffPool idea:**
  - **Hierarchically pool node embeddings**

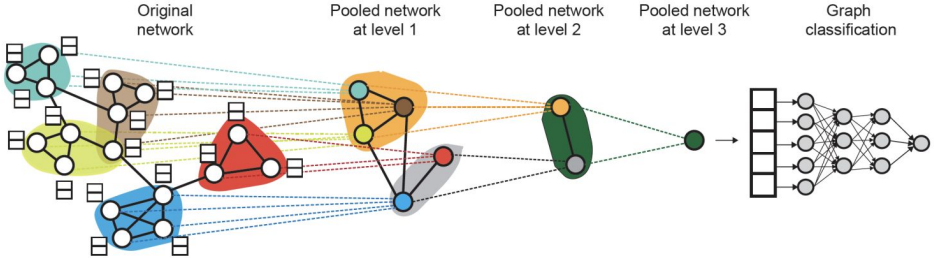


  - **Leverage 2 independent GNNs at each level**
    - **GNN A:** Compute node embeddings
    - **GNN B:** Compute the cluster that a node belongs to
  - **GNNs A and B at each level can be executed in parallel**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Hierarchical Graph Pooling – Diffpool

- **DiffPool idea:**



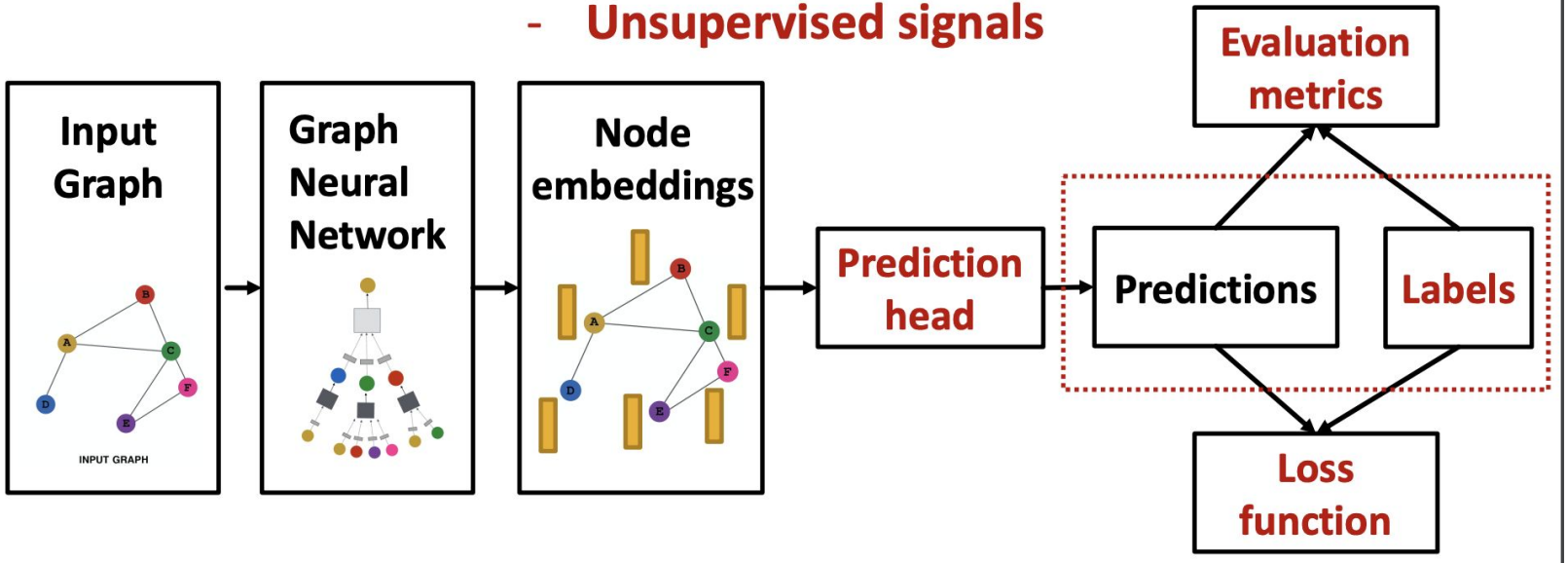
  - **For each Pooling layer**
    - Use clustering assignments from **GNN B** to aggregate node embeddings generated by **GNN A**
    - Create a **single new node** for each cluster, maintaining edges between clusters to generated a new **pooled** network
  - **Jointly train GNN A and GNN B**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Ground Truth

(2) Where does ground-truth come from?

- **Supervised labels**
- **Unsupervised signals**

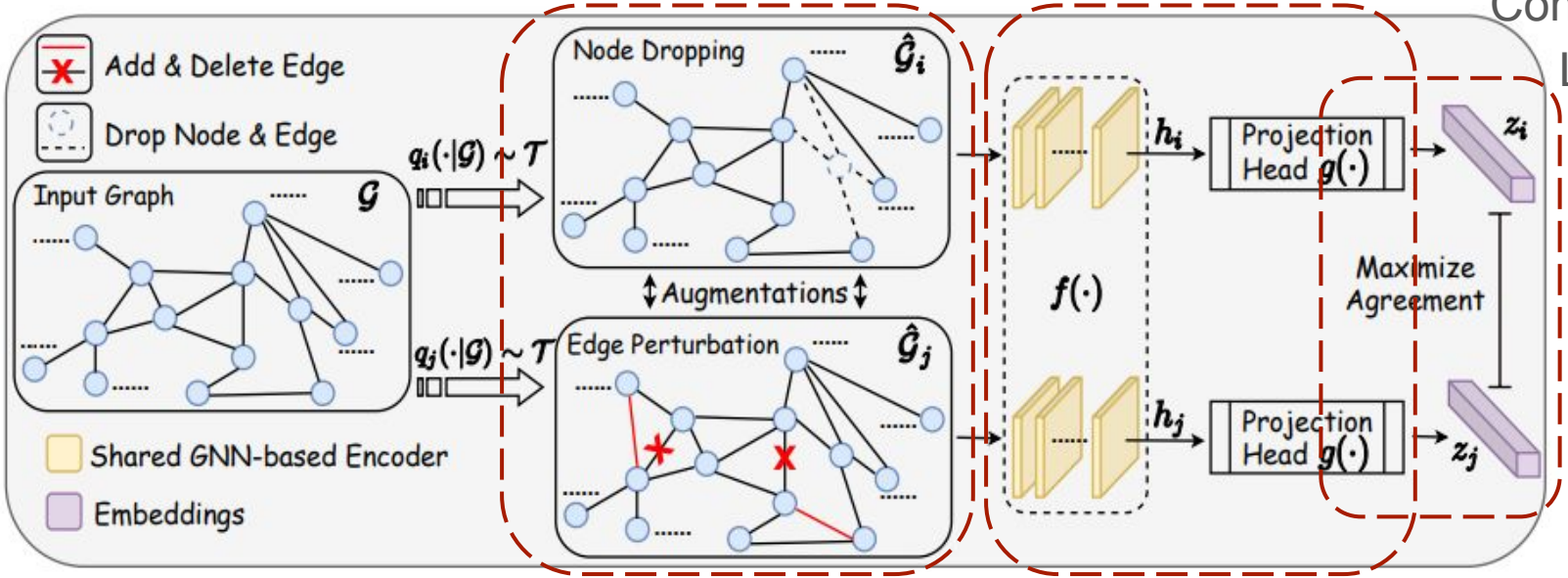


Self Supervised Learning in Graphs

- **Node-level y_v.** Node statistics: such as clustering coefficient, PageRank, …
- **Edge-level y_{uv}.** Link prediction: hide the edge between two nodes, predict if there should be a link
- **Graph-level y_G.** Graph statistics: for example, predic if two graphs are isomorphic

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Graph Contrastive Learning with Augmentations



$$\tilde{\mathcal{L}}_{\mathrm{GCL}}(f, q_1, q_2, \{G^{(i)}\}_{i=1}^m)$$
$$= -\frac{1}{m}\sum_{i=1}^{m} \mathrm{sim}\left(f(q_1(G^{(i)})), f(q_2(G^{(i)}))\right)$$

Augmented Graph Views

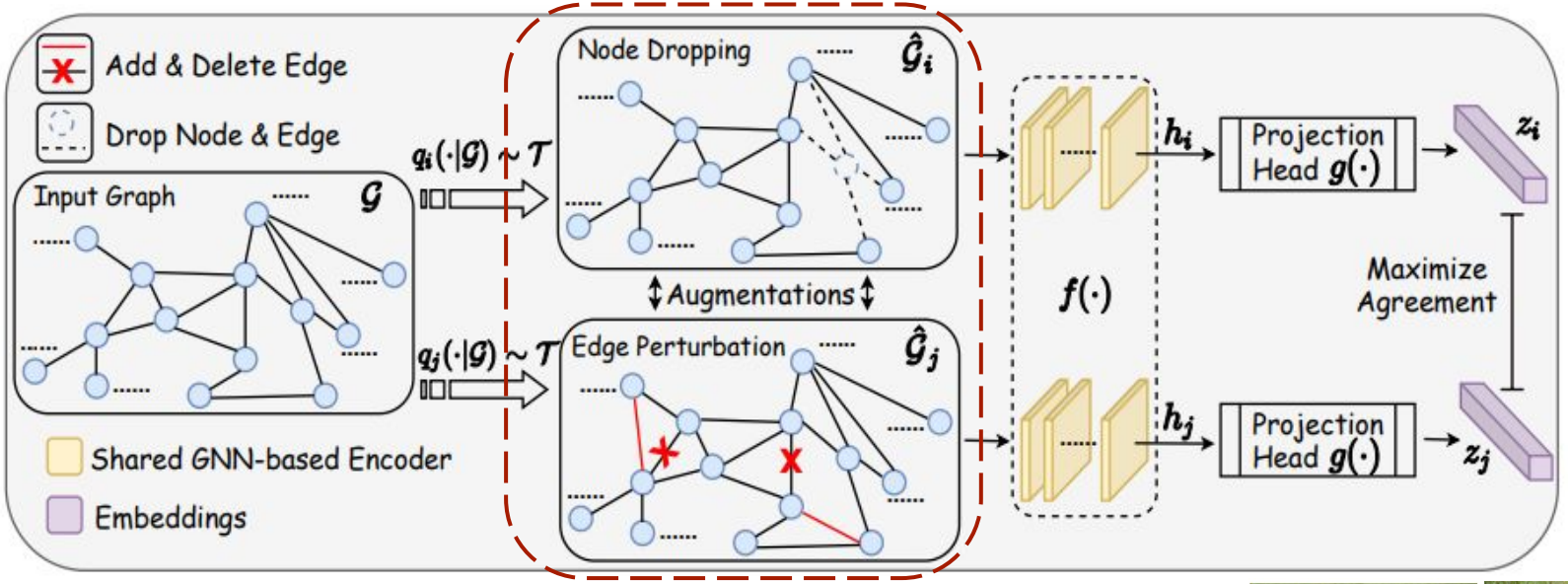
Shared-Weight GNN

*You et al., Graph Contrastive Learning with Augmentations (**GraphCL**), NeurIPS'20*

$q_1 : \mathbb{G} \to \mathbb{G}, q_2 : \mathbb{G} \to \mathbb{G}$

Optimizing lower bound of mutual information between graph views

Graph Contrastive Learning with Augmentations

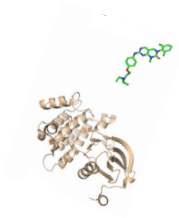


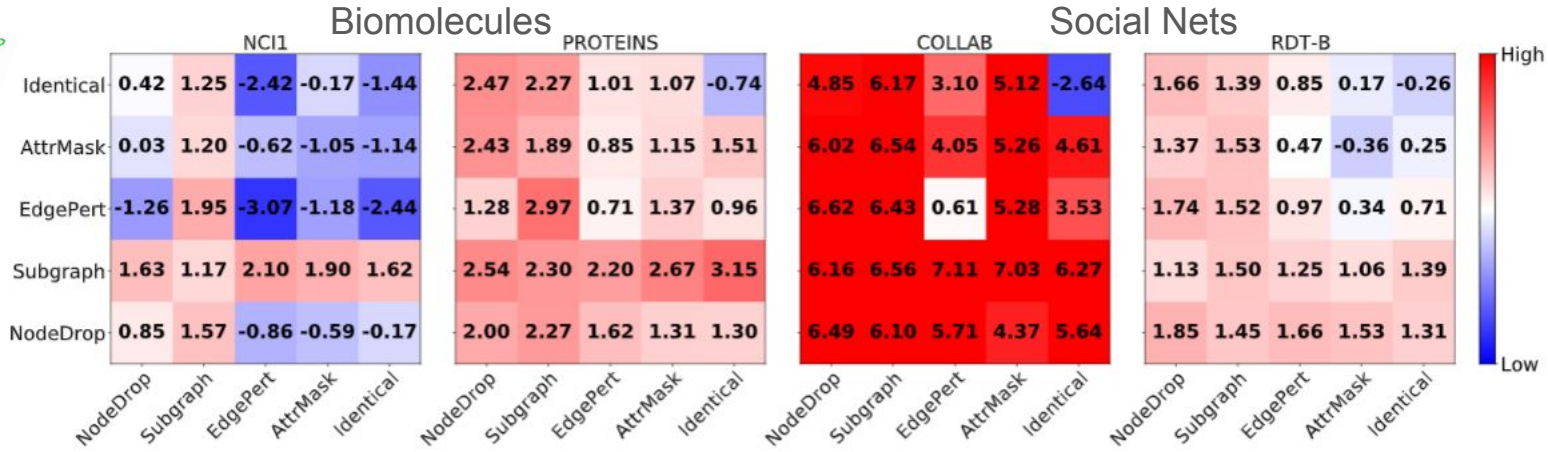
$$\tilde{\mathcal{L}}_{\mathrm{GCL}}(f, q_1, q_2, \{G^{(i)}\}_{i=1}^m)$$
$$= -\frac{1}{m}\sum_{i=1}^{m} \mathrm{sim}\Big(f(q_1(G^{(i)})), f(q_2(G^{(i)}))\Big)$$
$$q_1 : \mathbb{G} \to \mathbb{G}, q_2 : \mathbb{G} \to \mathbb{G}$$

Augmented Graph Views

Invariance Prior

The Choice of Graph Augmentations in GraphCL





Datasets	Category	Graph Num.	Avg. Node	Avg. Degree
NCI1	Biochemical Molecules	4110	29.87	1.08
PROTEINS	Biochemical Molecules	1113	39.06	1.86
COLLAB	Social Networks	5000	74.49	32.99
RDT-B	Social Networks	2000	429.63	1.15

Caltech EAS AI BOOTCAMP

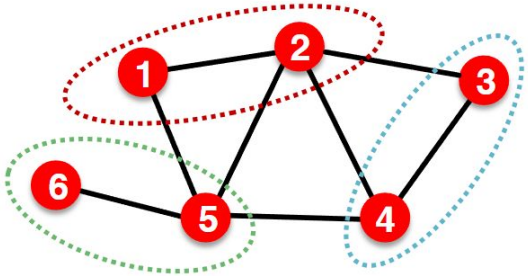
Why Splitting Graphs is Special

- **Node classification: Each data point is a node**
- Here **data points are NOT independent**
  - **Node 5 will affect our prediction on node 1,** because it will participate in message passing → affect node 1's embedding

Training

Validation

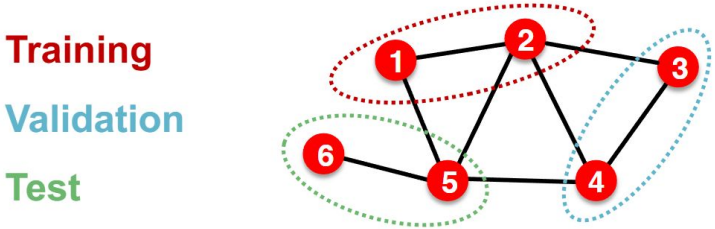
Test



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Transductive Setting

- **Solution 1 (Transductive setting): The input graph can be observed in all the dataset splits (training, validation and test set).**
- **We will only split the (node) labels**
  - **At training time,** we compute embeddings **using the entire graph**, and train **using node 1&2's labels**
  - **At validation time,** we compute embeddings **using the entire graph**, and **evaluate on node 3&4's labels**



Taken from Stanford CS224W course: http://cs244w.stanford.edu

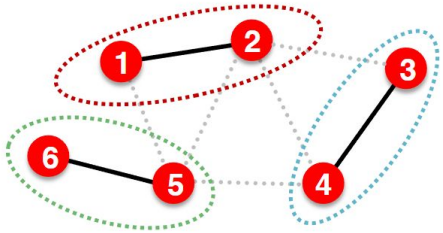
Inductive Setting

- **Solution 2 (Inductive setting): We break the edges between splits to get multiple graphs**
  - **Now we have 3 graphs that are independent. Node 5 will not affect our prediction on node 1 any more**
  - **At training time,** we compute embeddings **using the graph over node 1&2**, and train **using node 1&2's labels**
  - **At validation time,** we compute embeddings **using the graph over node 3&4**, and **evaluate on node 3&4's labels**

Training

Validation

Test



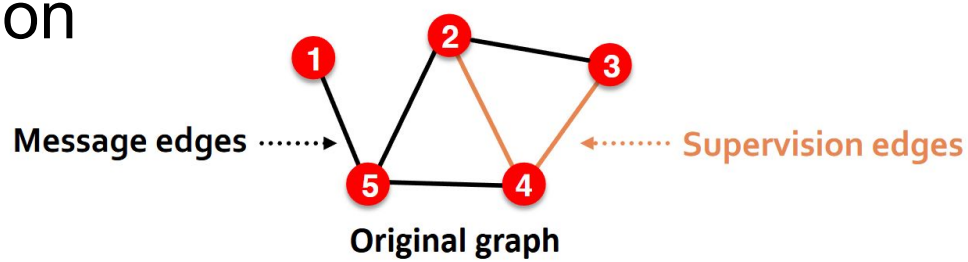
Taken from Stanford CS224W course: http://cs244w.stanford.edu

Transductive / Inductive Settings

- **Transductive setting:** training / validation / test sets are **on the same graph**
  - 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
- **Inductive setting:** 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

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Link Prediction

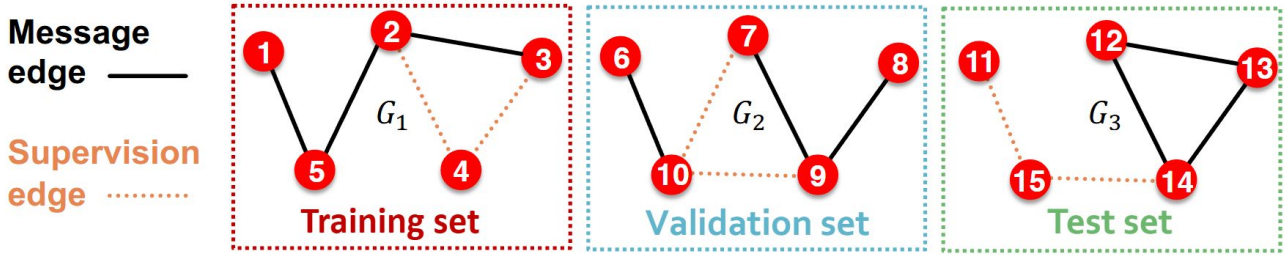


- **For link prediction, we will split edges twice**
- **Step 1: Assign 2 types of edges in the original graph**
  - **Message edges: Used for GNN message passing**
  - **Supervision edges: Use for computing objectives**
  - **After step 1:**
    - **Only message edges will remain in the graph**
    - **Supervision edges are used as supervision for edge predictions made by the model, will not be fed into GNN!**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Link Prediction

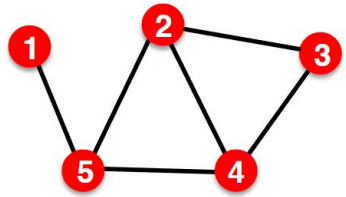
- **Step 2: Split edges into train / validation / test**
- **Option 1: Inductive link prediction split**
  - **Suppose we have a dataset of 3 graphs. Each inductive split will contain an independent graph**
  - **In train or val or test set, each graph will have 2 types of edges: message edges + supervision edges**
    - **Supervision edges** are not the input to GNN



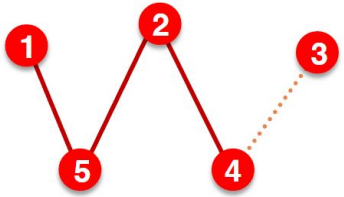
Taken from Stanford CS224W course: http://cs244w.stanford.edu

Link Prediction

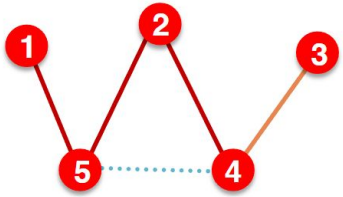
- **Option 2: Transductive link prediction split:**



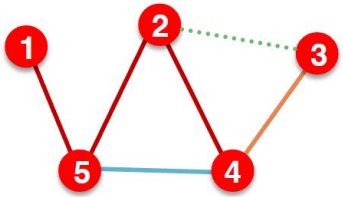
The original graph



(1) At training time:
Use **training message edges** to predict **training supervision edges**



(2) At validation time:
Use **training message edges & training supervision edges** to predict **validation edges**



(3) At test time:
Use **training message edges & training supervision edges & validation edges** to predict **test edges**

Taken from Stanford CS224W course: http://cs244w.stanford.edu