Chapter 4. Training GNNs

Overview

A. Stacking GNN layers

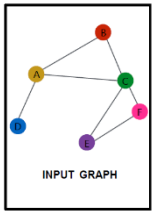
B. Prediction with GNNs

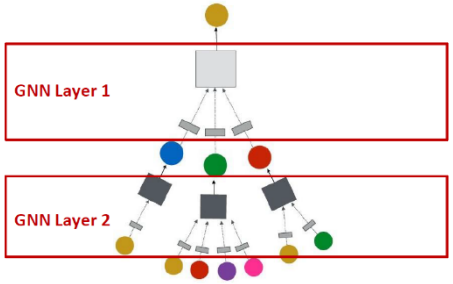
C. Training GNNs

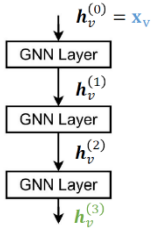
D. GNN prediction tasks

A. Stacking GNN Layers

- Stack layers sequentially
- Adding skip connections







많은 층을 쌓으면, 층 수만큼의 hop수의 neighbor 정보를 모아올 수 있지만, Over smoothing 문제가 발생하기도 한다.

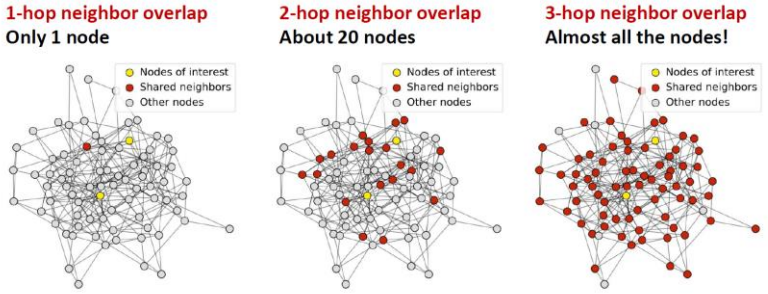
Over-Smoothing: All node embeddings converge to the same value

1) Over Smoothing 발생 이유

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 overlap for two nodes

- The shared neighbors quickly grows when we increase the number of hops (number of GNN layers)



두 노드의 receptive field가 겹치면 GNN layer의 수(hop수)가 매우 빠르게 커지는 경향이 있다.

- Over-smoothing can be explained via receptive field.

a. The embedding of a node is determined by its receptive field.

b. If two nodes have highly-overlapped receptive fields, their embeddings are highly similar.

c. Stack many GNN layers

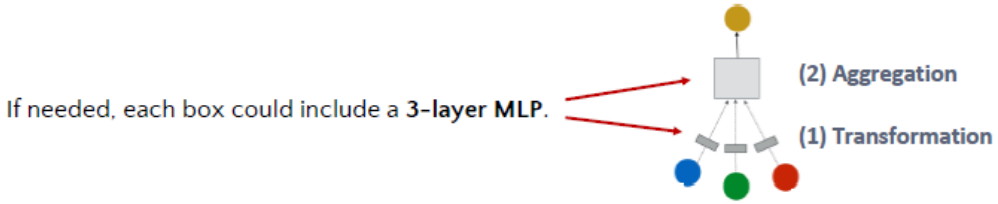
→ nodes will have highly-overlapped receptive fields

→ node embeddings will be highly similar

→ suffer from the over-smoothing problem.

2) How to overcome the Over-Smoothing

(1) Solution 1: increase the expressive power within each GNN layer



Each transformation or aggregation function only include one linear layer.

(각각의 임베딩하고 취합하는 함수가 오직 하나의 linear layer를 포함하게 만듬.)

Make aggregation / transformation become a deep neural network

(두 함수가 deep NN이 되게 만듬.)

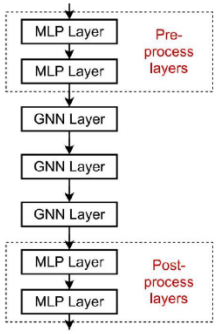
(2) Solution 2: Message를 전달하지 않는 layer를 중간에 만들어라

a. GNN 모델은 반드시 GNN layer만을 필요로 X

b. GNN layer 사이에 MLP layer를 추가해도 됨.

- **Pre-processing layers:**
  encodes node features
  - When nodes represent images/text.
- **Post-processing layers:**
  reasoning / transformation over node embeddings are needed
  - Graph classification, knowledge graphs

– In practice, adding these layers works great.



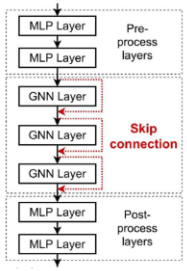
3) Skip Connection <- 만약 GNNs Layer를 필요로 한다면?

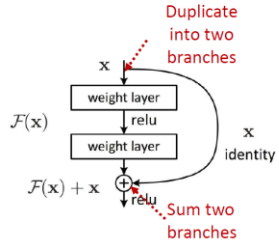
a. Skip Connection !!

b. 임베딩 결과가 원래 정보와 완전 상이할 수 있음

c. 이 경우 Skip Connection을 통해 두 정보를 더해주면 됨.(Shortcuts)

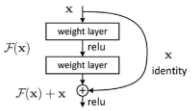
- Skip connections
  – Before adding shortcuts: **F(x)**
  – After adding shortcuts: **F(x) + x**





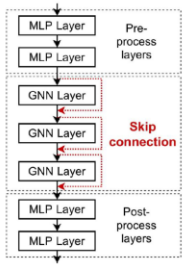
Standard GCN layer: F(x)

$$h_v^{(l)} = \sigma\left(\Sigma_{u \in N(v)} W^{(l)} \frac{h_u^{(l-1)}}{|N(v)|}\right)$$



GCN layer with skip connection: F(x) + x

$$h_v^{(l)} = \sigma\left(\Sigma_{u \in N(v)} W^{(l)} \frac{h_u^{(l-1)}}{|N(v)|} + h_v^{(l-1)}\right)$$



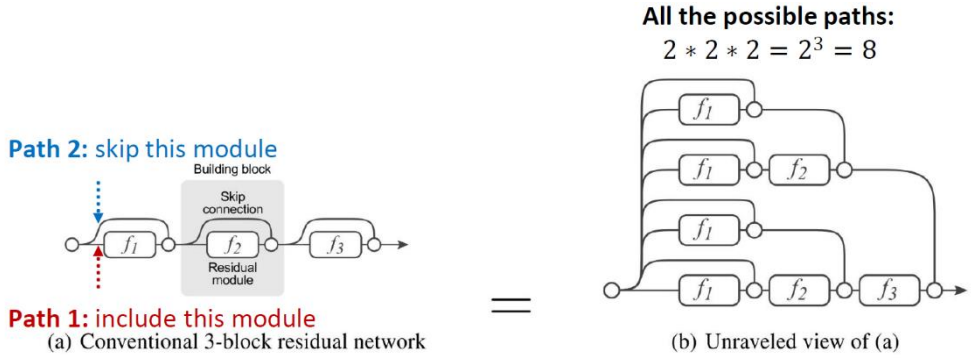
d. Skip Connection은 결국 mixture model을 만들어 내는 것과 같다.

e. N개의 Skip Connection은 2^N개의 path의 경우의 수를 가짐

f. 자연스럽게 shallow GNN과 deep GNN의 mixture model을 가지게 됨.

4) Skep Connection

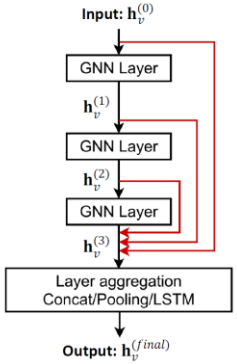
Skip Connection을 N개의 모듈로 이용하는 것



5) Other Skip Connection: Directly Skip to the last layer

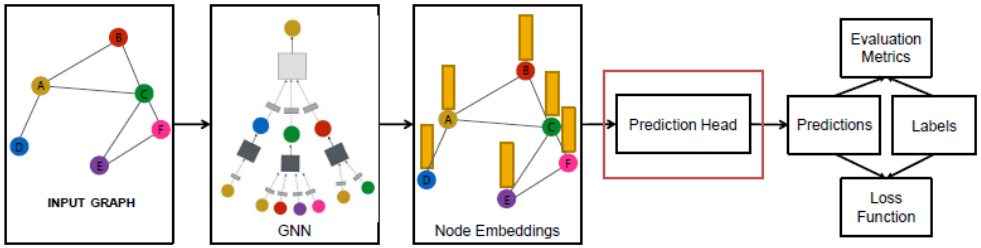
Directly skip to the last layer

- The final layer directly aggregates from the all the node embeddings in the previous layers.



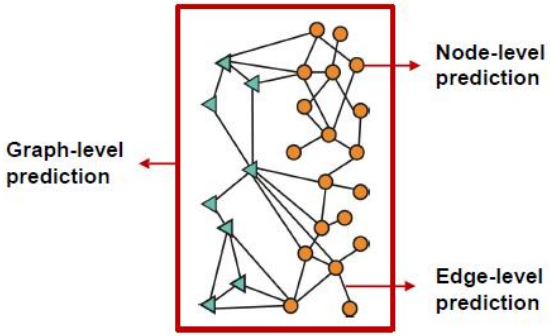
B. GNN Prediction

1) GNN Training Pipeline



GNN의 출력 값은 노드 임베딩의 집합이다. (Set of node embeddings $\{h_v^{(L)} \forall v \in G\}$)

GNN을 통해 풀 수 있는 문제는 크게 3 Level로 나눠진다.



2) Node Level Prediction

GNN 모델 학습 이후 d- 차원의 Node embedding값을 얻는다. 노드 임베딩을 이용해 direct하게 예측한다.

(1) K-Way prediction

a. Classification: Classify among k categories

b. Regression: regress on k targets

(2) $y_v = Head_{node}\left(h_v^{(L)}\right) = W^{(H)} h_v^{(L)}$

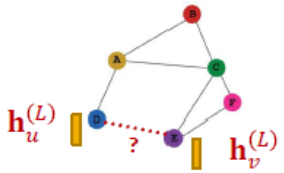
a. $W^{(H)} \in R^k$: Node embedding을 mapping($h_v^{(L)}$ -> y_v) -> Loss 계산 가능

3) Edge Level Prediction heads

a. Node embedding pair를 통해 prediction

(1) K-Way Prediction

a. $y_{uv} = Head_{edge}\left(h_v^{(L)}, h_u^{(L)}\right)$

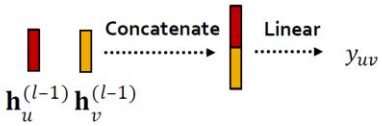


(2) 두 임베딩을 어떻게 사용하느냐?

a. Concatenation + Linear

Concatenation + Linear

- As in graph attention



- $y_{uv} = Linear(Concat(h_u^{(L)}, h_v^{(L)}))$
- $Linear(\cdot)$ maps the concatenated embeddings to k-dim embeddings (k-way prediction)

b. Dot Product

Dot product

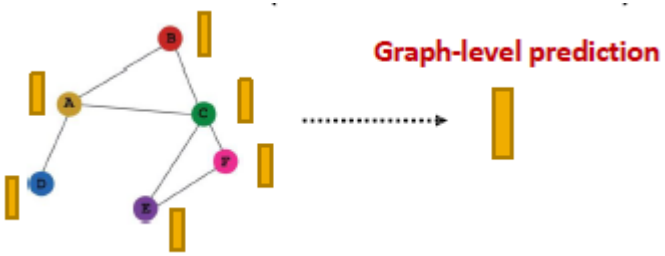
- $y_{uv} = \left(h_u^{(L)}\right)^T \cdot h_v^{(L)}$
- Only applies to 1-way prediction (e.g., link prediction: predict the existence of an edge)
- Applying k-way prediction:
  - Similar to multi-head attention: $W^{(1)}, \dots, W^{(k)}$ trainable
  - $y_{uv}^{(1)} = \left(h_u^{(L)}\right)^T W^{(1)} h_v^{(L)}$
  - ⋮
  - $y_{uv}^{(k)} = \left(h_u^{(L)}\right)^T W^{(k)} h_v^{(L)}$
  - $y_{uv} = Concat\left(y_{uv}^{(1)}, \dots, y_{uv}^{(k)}\right) \in \mathbb{R}^k$

4) Graph Level Prediction

a. 모든 Node Embedding을 이용해 Prediction 진행

(1) K-Way prediction

a. $y_v = Head_{graph}\left(\{h_v^{(L)} \in R^d, \forall\, v \in G\}\right)$



b. $Head_{graph}(\cdot)$ 는 GNN layer의 $AGG(\cdot)$ 와 동일하다.

c. Global mean/max/sum pooling을 이용함.

- $Head_{graph}\left(\{h_v^{(L)} \in \mathbb{R}^d, \forall v \in G\}\right)$

1. Global mean pooling

$$y_G = Mean\left(\{h_v^{(L)} \in \mathbb{R}^d, \forall v \in G\}\right)$$

2. Global max pooling

$$y_G = Max\left(\{h_v^{(L)} \in \mathbb{R}^d, \forall v \in G\}\right)$$

3. Global sum pooling

$$y_G = Sum\left(\{h_v^{(L)} \in \mathbb{R}^d, \forall v \in G\}\right)$$

d. 하지만 이 방법은 Small-Size Graph에서만 좋음

Large Size graph에서는 다른 방식을 사용함. 계층적 mean pooling, Hierarchical Global Pooling 라는 방식을 사용. 모든 노드의 임베딩을 계층적으로 취합하는 것이다.

ReLU(Sum())을 통해 취합: 첫 두 노드와 마지막 두 노드들을 각각 취합한 후, 층을 올려가며 반복적으로 진행.

5) Issue of Global Pooling

Global Pooling은 Large-Scale Graph에 적용하면 정보 손실(Lose information)을 유발한다. GNN에서 Global Pooling을 하는 이유는 출력값으로 나오는 임베딩의 크기를 고정시켜주기 위함이다. 하지만, 이 과정에서 mean, max, sum같은 연산을 취하는데 그렇게 할 경우 노드 수가 많은 Large-Scaled Graph에서 문제가 발생할 수 있다.

Ex)

G1 -> {-5,0,2,3} -> Sum -> 0

G2 -> {-50,0,20,30} -> Sum -> 0

두 그래프의 구조가 다르지만 pooling값이 0으로 같아질 수 있다.

6) Hierarchical Pooling in Practice

모든 Node embedding 값을 hierarchical하게 aggregation함.

Ex)

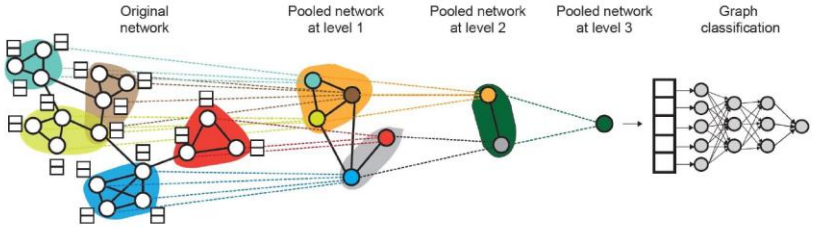
ReLU(Sum(으)): 처음에 첫 두 노드와 마지막 두 노드를 각자 aggregate하고, 이후 반복하여 최종적으로 prediction을 진행

1. $y_a = ReLU(Sum(\{-5, 0\})) = 0$, $y_b = ReLU(Sum(\{2, 3\})) = 5$
2. $y_{G1} = ReLU(Sum(\{y_a, y_b\}) = 5$
   - Node embeddings of $G2$: {-50, 0, 20, 30}

1. $y_a = ReLU(Sum(\{-50, 0\})) = 0$, $y_b = ReLU(Sum(\{20, 30\}))= 50$
2. $y_{G1} = ReLU(Sum(\{y_a, y_b\})) = 50$
   - Now we can differentiate $G1$ and $G2$

7) Hierarchical Pooling in Practice

DiffPool: **hierarchically pool node embeddings**



Ying et al. Hierarchical Graph Representation Learning with Differentiable Pooling, NeurIPS 2018

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**

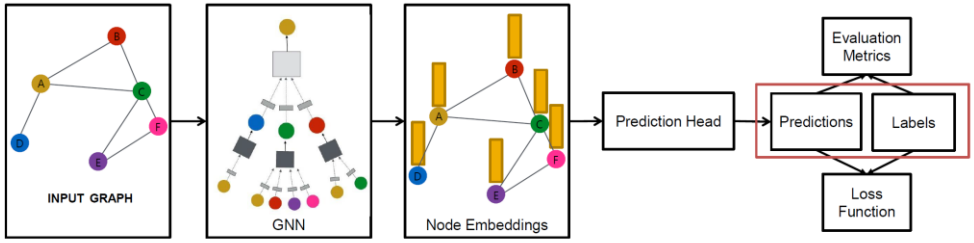
For each pooling layer

- 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 generate a new pooled network

Jointly train GNN A and GNN B

C. Training GNNs

- Where does ground-truth come from?
  - Supervised labels
  - Unsupervised signals



1) Supervised vs Unsupervised

A. Supervised

a. Labels come from external sources

B. Unsupervised(self-supervised)

a. Signals come from external sources

2) Loss Function

(1) Classification Loss: Cross-Entropy

Cross entropy (CE) is a common loss function in classification

K-way prediction for i-th data point:

$$CE\left(y^{(i)}, \hat{y}^{(i)}\right) = -\Sigma_{j=1}^{K} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$$

where

$y^{(i)} \in \mathbb{R}^K$: one-hot label encoding | 0 | 0 | 1 | 0 | 0 |

$\hat{y}^{(i)} \in \mathbb{R}^K$: prediction after $Softmax(\cdot)$ | 0.3 | 0.2 | 0.9 | 0.1 | 0.2 |

Total loss over all N training examples

$$Loss = \Sigma_{i=1}^{N} CE\left(y^{(i)}, \hat{y}^{(i)}\right)$$

(2) Regression Loss: MSE

Mean squared error (i.e., L2 loss) is often used in regression tasks.

K-way regression for i-th data point:

$$MSE\left(y^{(i)}, \hat{y}^{(i)}\right) = \Sigma_{j=1}^{K} \left(y_j^{(i)} - \hat{y}_j^{(i)}\right)^2$$

where

$y^{(i)} \in \mathbb{R}^K$: real-valued vector of targets | 2.1 | 0.6 | 1.1 | 0.8 | 0.3 |

$\hat{y}^{(i)} \in \mathbb{R}^K$: real-valued vector of predictions | 1.8 | 0.2 | 0.9 | 0.7 | 0.2 |

Total loss over all N training examples

$$Loss = \Sigma_{i=1}^{N} MSE\left(y^{(i)}, \hat{y}^{(i)}\right)$$

3) Evaluation Metric

(1) Regression

a. Root mean square error (RMSE)

$$\sqrt{\frac{\Sigma_{i=1}^{N} \left(y_j^{(i)} - \hat{y}_j^{(i)}\right)^2}{N}}$$

b. Mean Absolute Error(MAE)

$$\frac{\Sigma_{i=1}^{N} \left|y_j^{(i)} - \hat{y}_j^{(i)}\right|}{N}$$

(2) Classification

a. Multi-Class Classification: Accuracy(Acc)

$$\frac{\mathbb{I}\left[argmax\left(\hat{\boldsymbol{y}}^{(i)}\right) = \boldsymbol{y}^{(i)}\right]}{N}$$

b. Binary Classification: Acc, Precision Recall

c. Metric agnostic to classification threshold: ROC AOC

$$\frac{\Sigma_{i=1}^{N} \left|y_j^{(i)} - \hat{y}_j^{(i)}\right|}{N}$$

Binary Classification Metrics

- Accuracy

$$\frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{|Dataset|}$$

- Precision (P)

$$\frac{TP}{TP + FP}$$

- Recall (R)

$$\frac{TP}{TP + FN}$$

- F1-Score

$$\frac{2P \cdot R}{P + R}$$

Confusion matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

ROC curve: captures the tradeoff in TPR and FPR **as the classification threshold is varied or a binary classifier**.

$$TPR = Recall = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

The dashed line represents performance of a random classifier

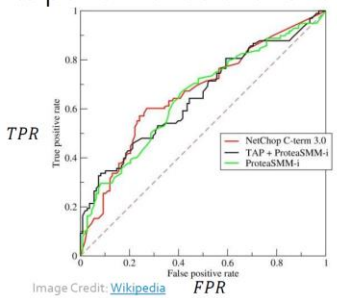


Image Credit: Wikipedia

ROC AUC: area under the ROC curve

The probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one

D. GNN Prediction Task

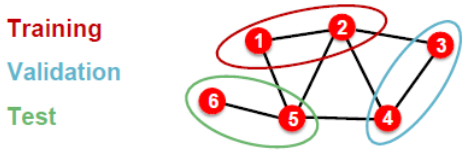
1) Inductive vs Transductive Setting

(1) Splitting Graph: Transductive Setting

Input graph can be observed in all the dataset splits (training, validation, and test sets)

Split only the node labels

- **At training time**, compute embeddings **using the entire graph**, and train **using nodes 1&2's labels**
- **At validation time**, compute embeddings **using the entire graph**, and **evaluate on nodes 3&4's labels**



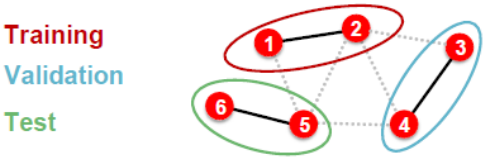
(2) Splitting Graph: Inductive Setting

Break the edges between splits **to get multiple graphs**

We have 3 graphs that are independent. Node 5 will not affect a prediction on node 1.

At training time, compute embeddings **using the graph over nodes 1&2**, and **train using nodes 1&2's labels**

At validation time, compute embeddings **using the graph over nodes 3&4**, and **evaluate on nodes 3&4's labels**



A. Transductive Setting

a. Train/Valid/Test set이 모두 같은 그래프에 있다.

b. Dataset이 결론적으로 하나의 그래프만 가짐.

c. 이 그래프를 3개로 나눠서 split.

d. Node/edge prediction task에만 사용 가능.

B. Inductive Setting

a. Train/Valid/Test set이 모두 다른 그래프

b. 전체 Dataset이 여러 그래프를 가짐.

c. Each split can only observe the graphs within the split. A successful model should generalize to unseen graphs

d. node / edge / graph tasks에 모두 이용 가능

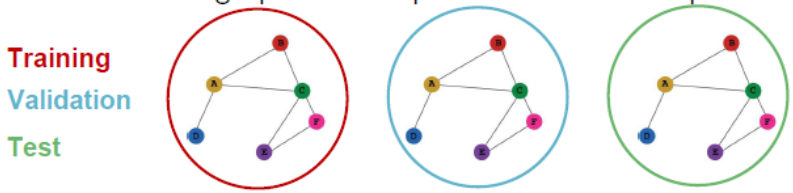
2) Node Classification

Transductive node classification

- All the splits can observe the entire graph structure, but can only observe the labels of their respective nodes

Inductive node classification

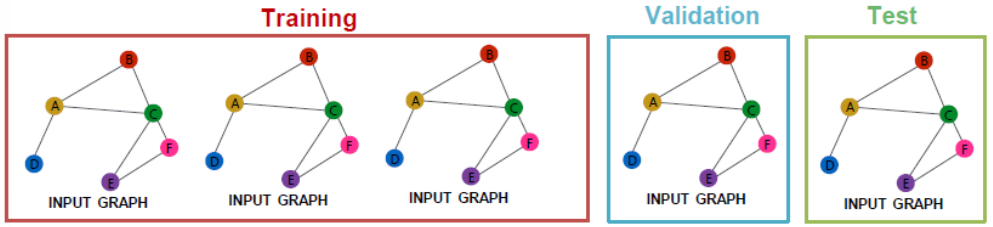
- For a dataset of 3 graphs, each split contains an independent graph



3) Graph Classification

Only the **inductive setting** is well defined for **graph classification**

- We have to test on unseen graphs
- For a dataset of 5 graphs, each split will contain independent graphs.

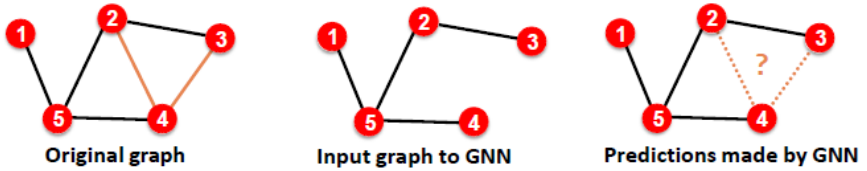


4) Link Prediction

Goal: predict missing edges

Setting up link prediction is tricky:

- Link prediction is an unsupervised / self-supervised task.
- We need to **create the labels** and **dataset splits** on our own.
  = We need to **hide some edges from the GNN** and **let the GNN predict if the edge exists**



5) Setting up Link Prediction

Link Prediction을 위해서는 Edge를 두 번 split한다.

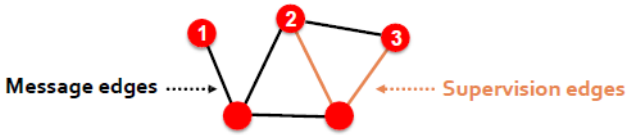
(1) Assign 2 types of edges in the original graph

a. Message Edges: for GNN message passing

b. Supervision Edges: for computing objectives

After Step 1, **only message edges will remain in the graph**

Supervision edges are used as supervision edge predictions made by the model, will not be fed into GNN.



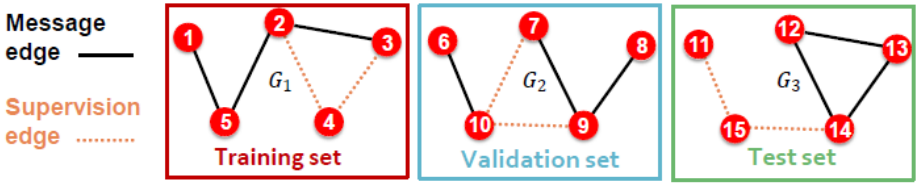
(2) Split edges into training / validation / test sets

a. Option 1: inductive link prediction split

b. Inductive setting에서는 3가지의 독립된 그래프로 split한다.

In training, validation, or test set, each graph will have 2 types of edges: **message edges** + supervision edges

- Supervision edges are not the input to GNN.



c. Option 2: Transductive link prediction split

d. 하나의 큰 그래프에서 Train / Valid / Test set을 나눈다.