Stanford CS224W: GNN Augmentation and Training

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

Announcements

- **Deadline extension for Colab 1 and project proposal**
  - Colab 1 due date extended to Tuesday, October 17
  - Project proposal due date extended to Tuesday, October 24
- **Homework 1**
  - Due Thursday, October 19 (1 week from now)
  - Recitation session on 1pm-3pm, Friday, October 13
    - Details on Ed
- Project information released on course website

Course project

- **Goal**: create long-lasting resources for your technical profiles + broader graph ML community
- Three types of projects
  - 1) Real-world applications of GNNs
  - 2) Tutorial on PyG functionality
  - 3) Implementation of cutting-edge research
- We will publish your blog posts on our course's [Medium page](#)!

1) Real-world applications of GNNs

- **Goal**: identify a specific use case and demonstrate how GNNs and PyG can be used to solve this problem
- **Output**: blog post, Google colab
- Example use cases
  - Fraud detection
  - Predicting drug interactions
  - Friend recommendation
- Check out the featured posts in the project information doc on the course website

2) Tutorial on PyG functionality

- **Goal**: develop a tutorial that explains how to use existing PyG functionality
- **Output**: blog post, Google colab
- Example topics for tutorials
  - PyG's explainability module
  - Methods for graph sampling (e.g., negative sampling, sampling on heterogeneous graphs)
  - Tutorial on GraphGym, a platform for designing and evaluating GNNs
- Check out example tutorials from PyG

3) Implementation of research

- **Goal**: implement interesting methods from a recent research paper in graph ML
- **Output**: PR to PyG [contrib](#), short blog post
- Project details

  - Implementation should include comprehensive testing and documentation on new functionality

  - Try to build on existing PyG and PyTorch code wherever possible

  - Note: this project is more manageable if you are already comfortable with PyTorch and deep learning. We also highly recommend group of 3.

Project logistics

- Project is worth 30% of your course grade
  - Project proposal (2 pages), due October 24
  - Project milestone, due November 9
  - Final reports, due December 14
- We recommend groups of 3, but groups of 2 are also allowed
  - Each group will be assigned a CA mentor who will provide feedback
- **Full project description is released on the [course website](http://cs224w.stanford.edu)!**

Stanford CS224W: GNN Augmentation and Training

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University

http://cs224w.stanford.edu
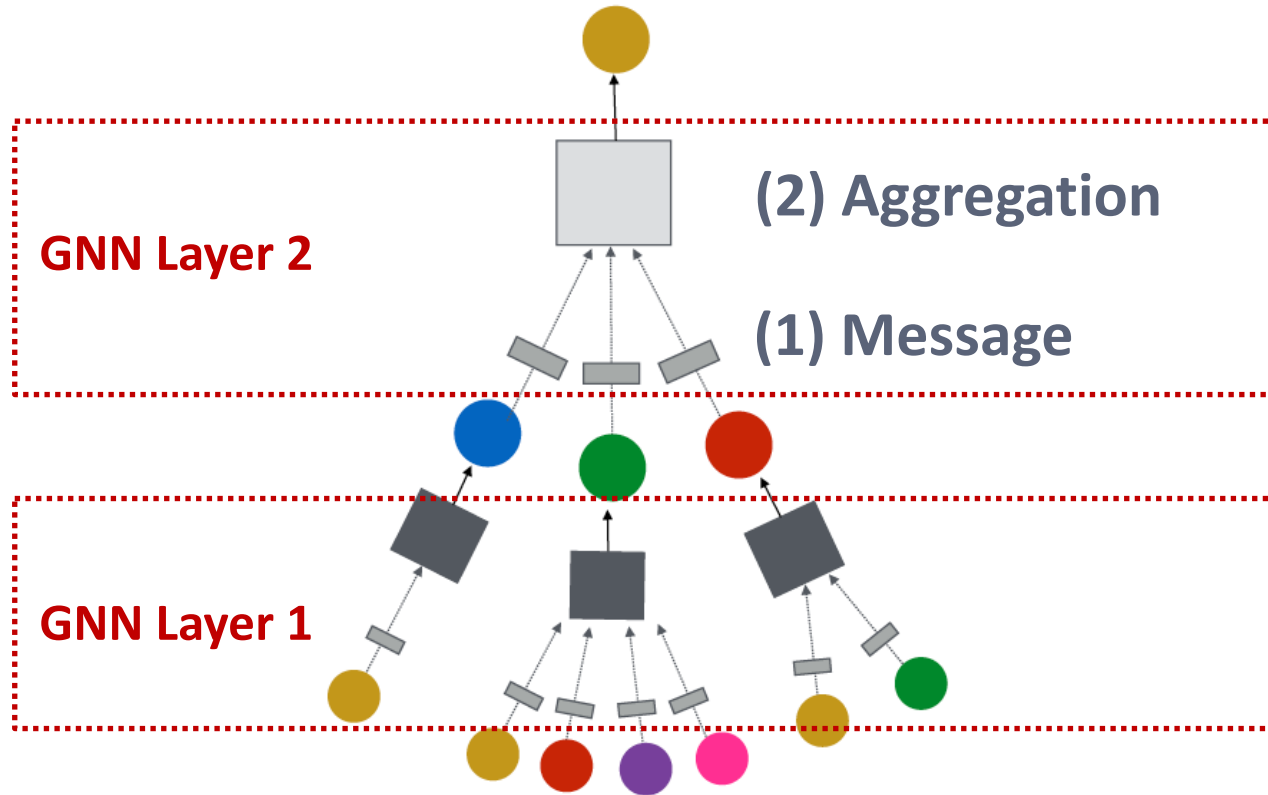
Recap: A General GNN Framework

TARGET NODE

INPUT GRAPH

(5) Learning objective
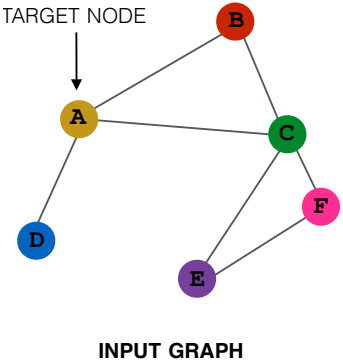
GNN Layer 2

(2) Aggregation

(1) Message

(3) Layer connectivity

GNN Layer 1

(4) Graph augmentation

Recap: A Single GNN Layer

- **Putting things together:**
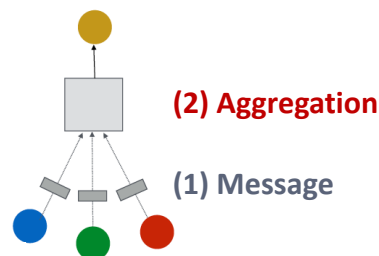
  - **(1) Message**: each node computes a message
    $$\mathbf{m}_u^{(l)} = \text{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), u \in \{N(v) \cup v\}$$

  - **(2) Aggregation**: aggregate messages from neighbors
    $$\mathbf{h}_v^{(l)} = \text{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}, \mathbf{m}_v^{(l)}\right)$$

  - **Nonlinearity (activation):** Adds expressiveness

    - Often written as $\sigma(\cdot)$: $\text{ReLU}(\cdot)$, $\text{Sigmoid}(\cdot)$, ...

    - Can be added to **message or aggregation**

(2) Aggregation

(1) Message

Recap: GNN Layer Connectivity

- **What if my problem still requires many GNN layers?**
- **Lesson 2: 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**
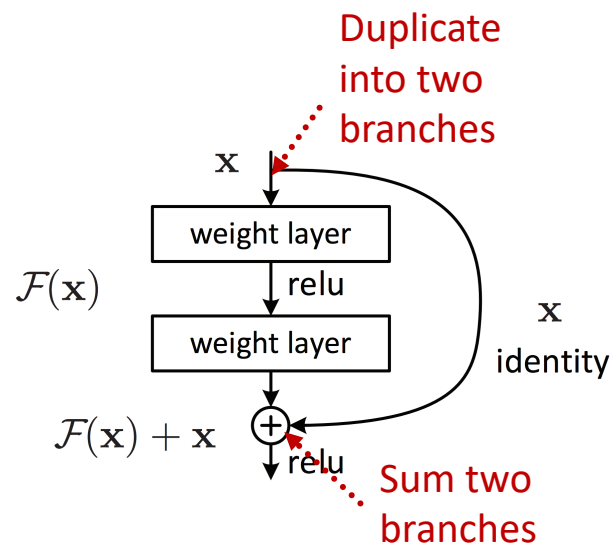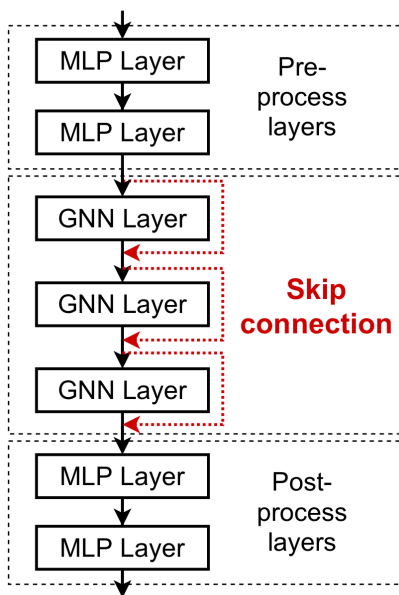
Idea of skip connections:
Before adding shortcuts:
$$F(\mathbf{x})$$
After adding shortcuts:
$$F(\mathbf{x}) + \mathbf{x}$$

Recap: Graph Manipulation

- **Graph Feature manipulation**

  - The input graph **lacks features** → **feature augmentation**

- **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**

    - Will cover later in lecture: Scaling up GNNs

Feature Augmentation on Graphs

Why do we need feature augmentation?

- **(2) Certain structures are hard to learn by GNN**
- Other commonly used augmented features:
  - **Degree distribution**
  - **Clustering coefficient**
  - **PageRank**
  - **Centrality**
  - **…**
- Any **feature we have introduced can be used!**

Add Virtual Nodes / Edges

- **Motivation:** Augment sparse graphs
- **(1) Add 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

Authors   **Papers**
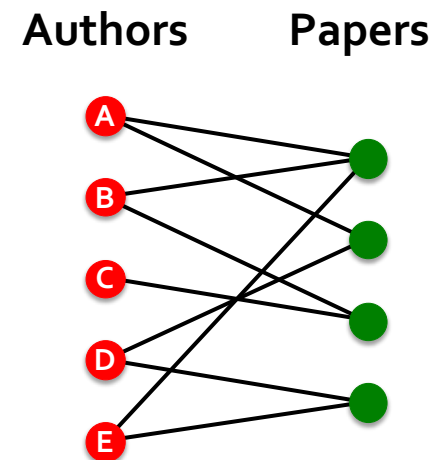
Add Virtual Nodes / Edges

- **Motivation:** Augment sparse graphs

- **(2) Add 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**

The virtual node

INPUT GRAPH
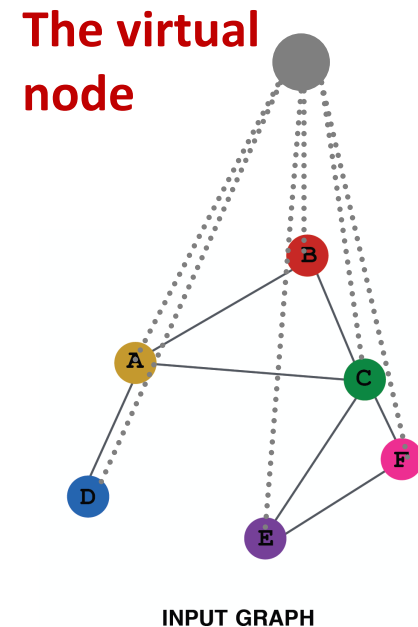
Node Neighborhood Sampling

- ## Previously:
  - ### All the nodes are used for message passing

TARGET NODE

INPUT GRAPH
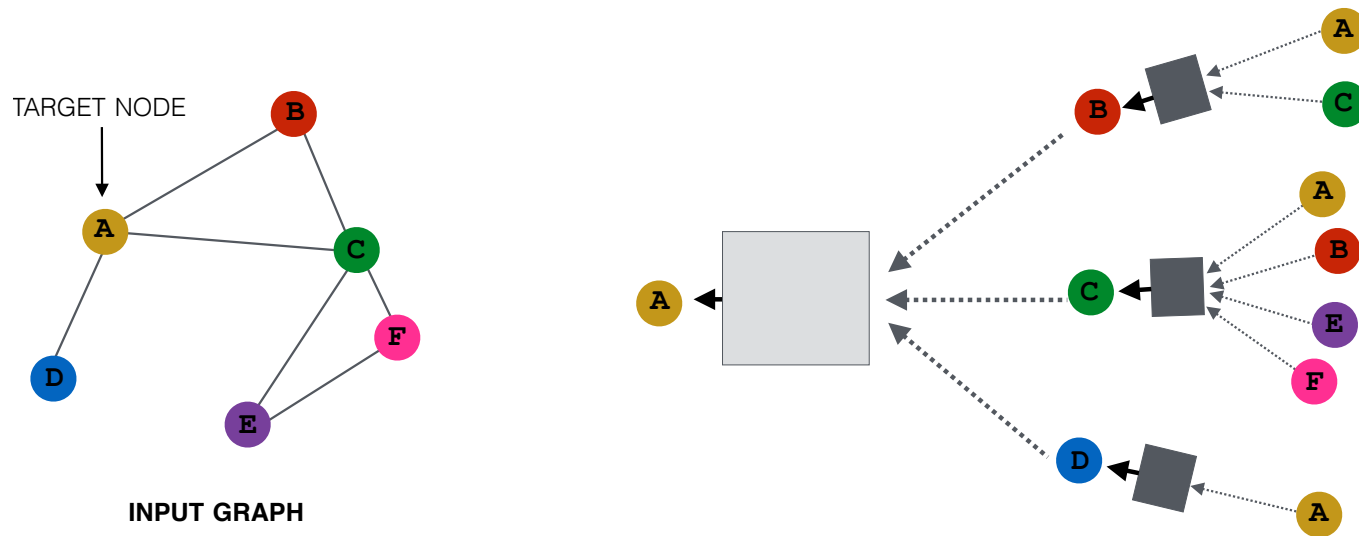
- ## New idea: (Randomly) sample a node's neighborhood for message passing

Neighborhood Sampling Example

- **For example, we can randomly choose 2 neighbors to pass messages**
  - Only nodes B and D will pass message to A


INPUT GRAPH
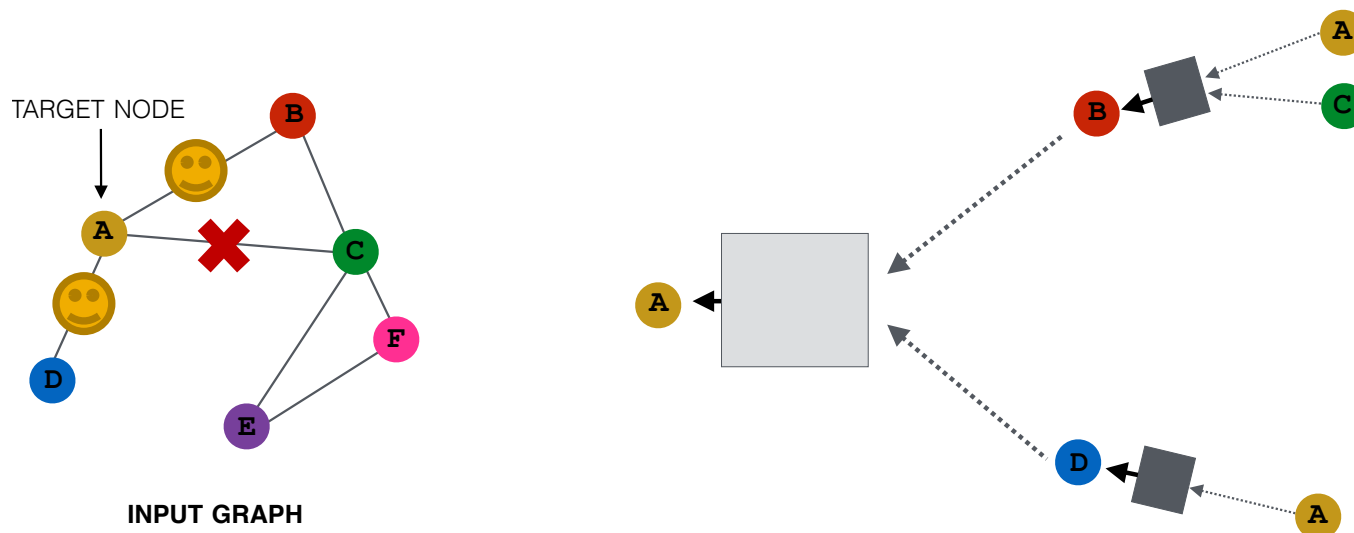
Neighborhood Sampling Example

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

INPUT GRAPH
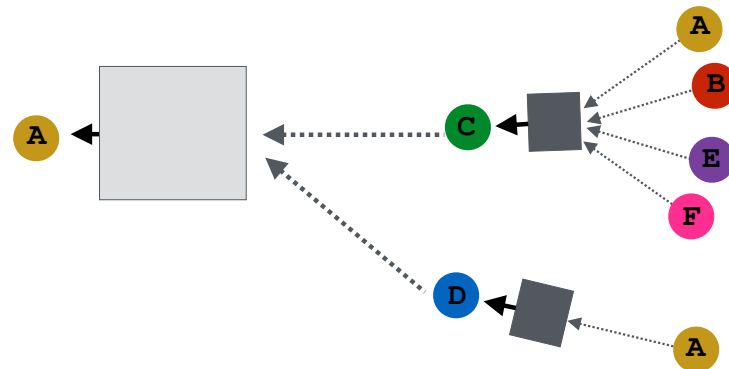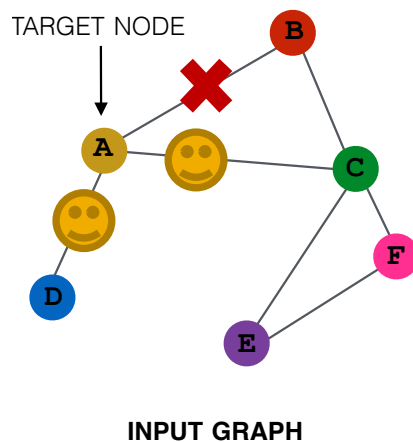
Neighborhood Sampling Example

- ## 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!

INPUT GRAPH
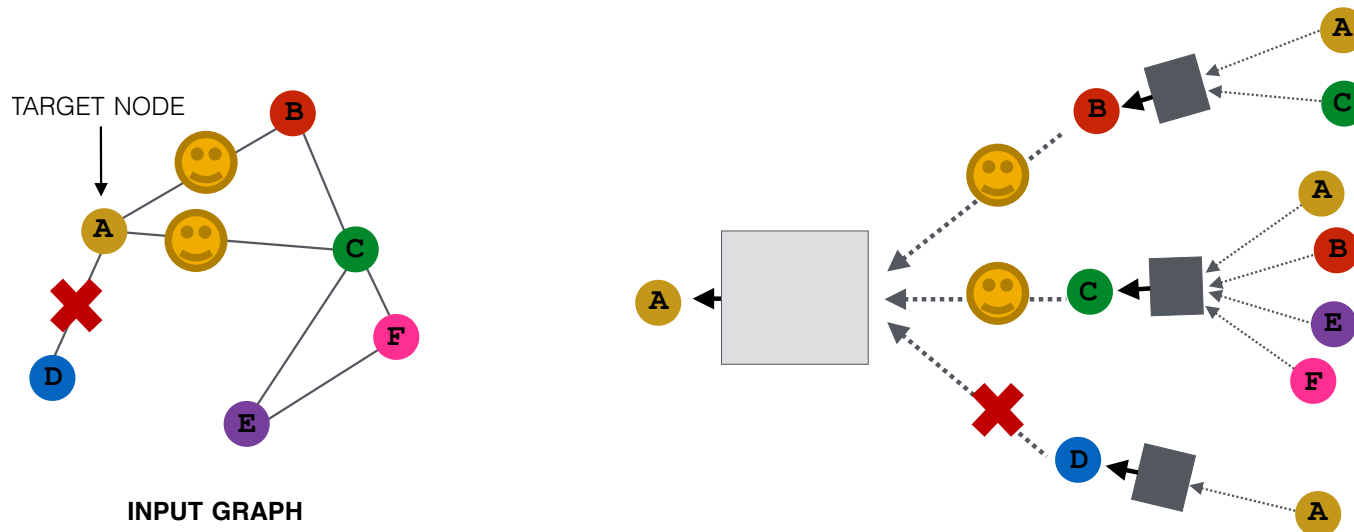
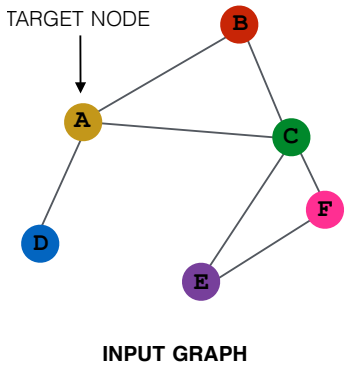# Stanford CS224W: Prediction with GNNs

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

A General GNN Framework (4)

TARGET NODE

INPUT GRAPH

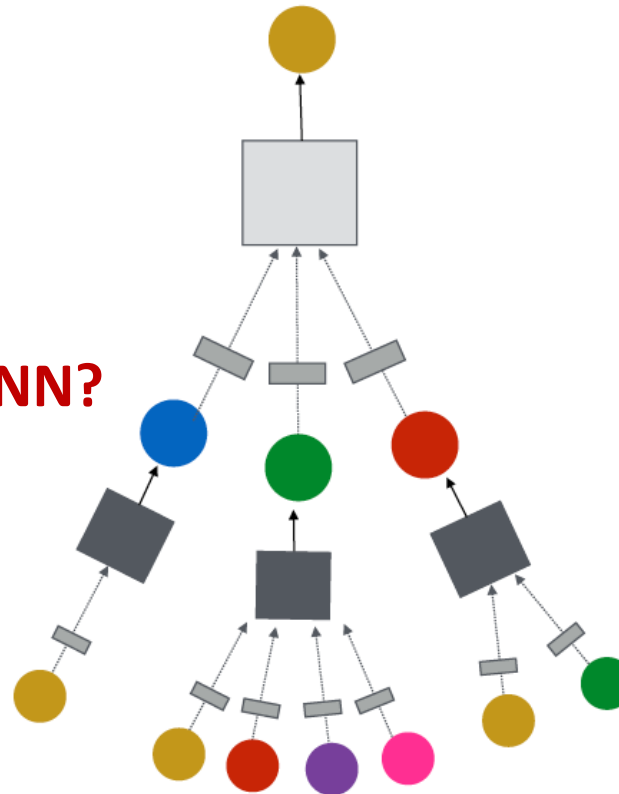**(5) Learning objective**

Next: How do we train a GNN?

GNN Training Pipeline

So far what we have covered
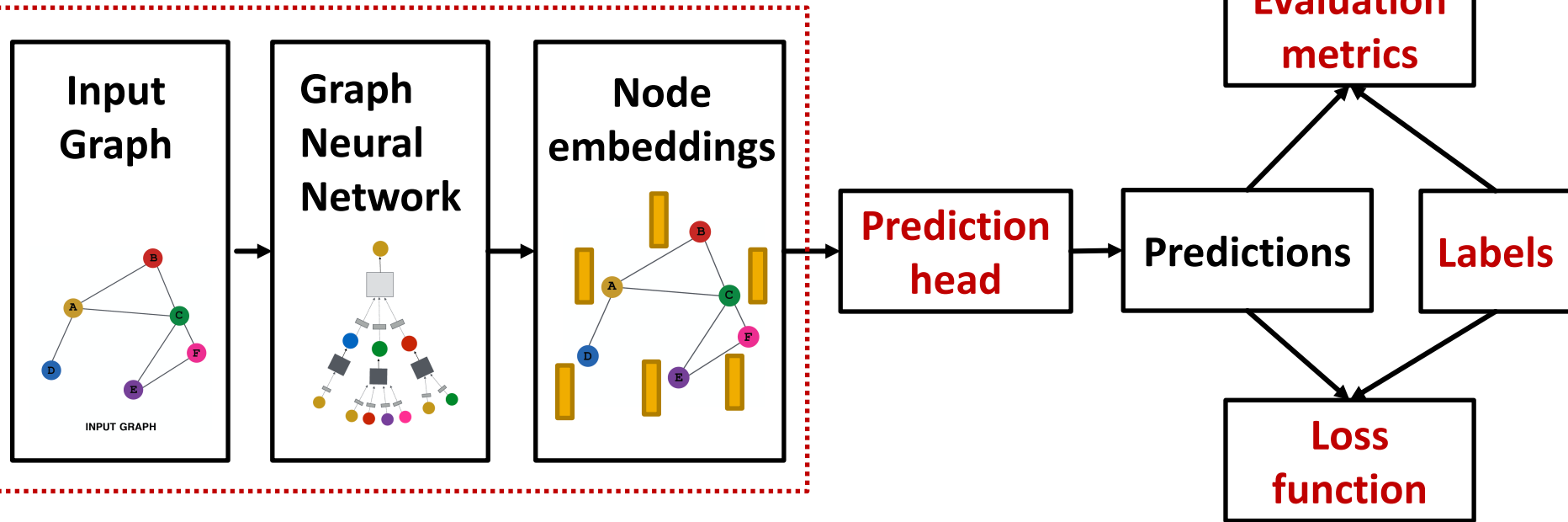
Output of a GNN: set of node embeddings
$$\{\mathbf{h}_v^{(L)}, \forall v \in G\}$$

GNN Training Pipeline (1)

(1) Different prediction heads:
- **Node-level tasks**
- **Edge-level tasks**
- **Graph-level tasks**

GNN Prediction Heads

- **Idea:** Different task levels require different prediction heads

Node-level prediction

Graph-level prediction

Edge-level prediction

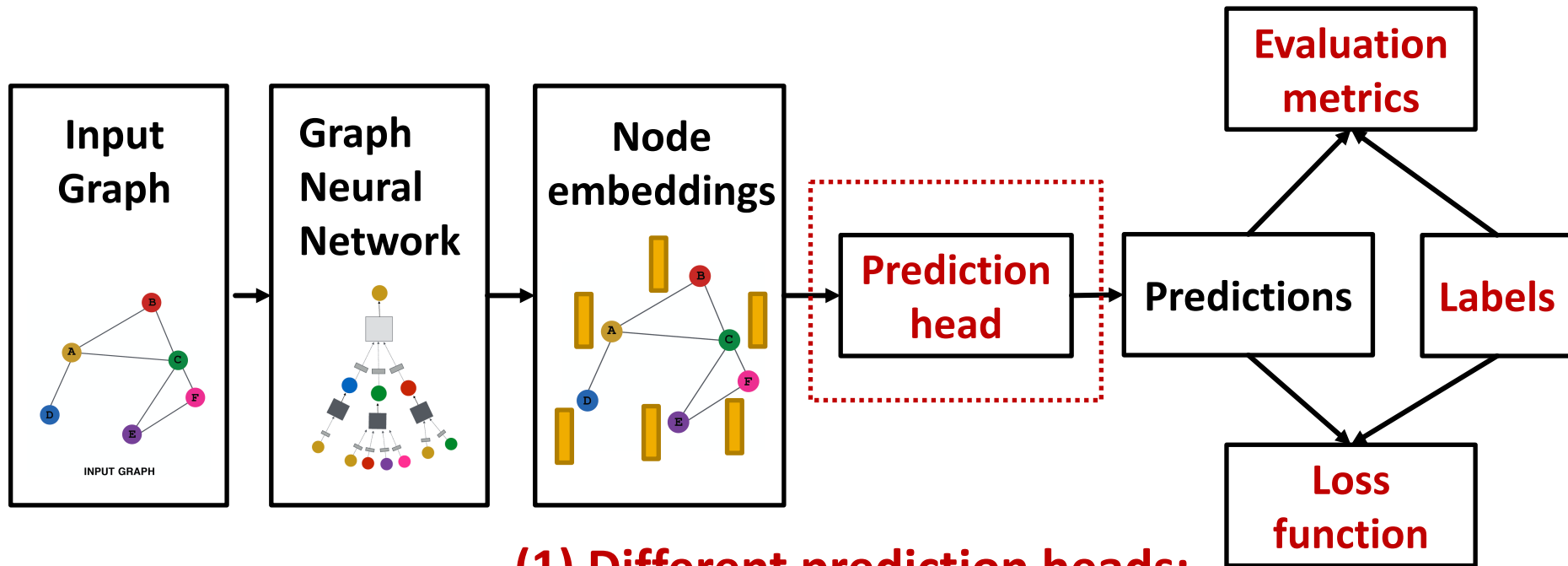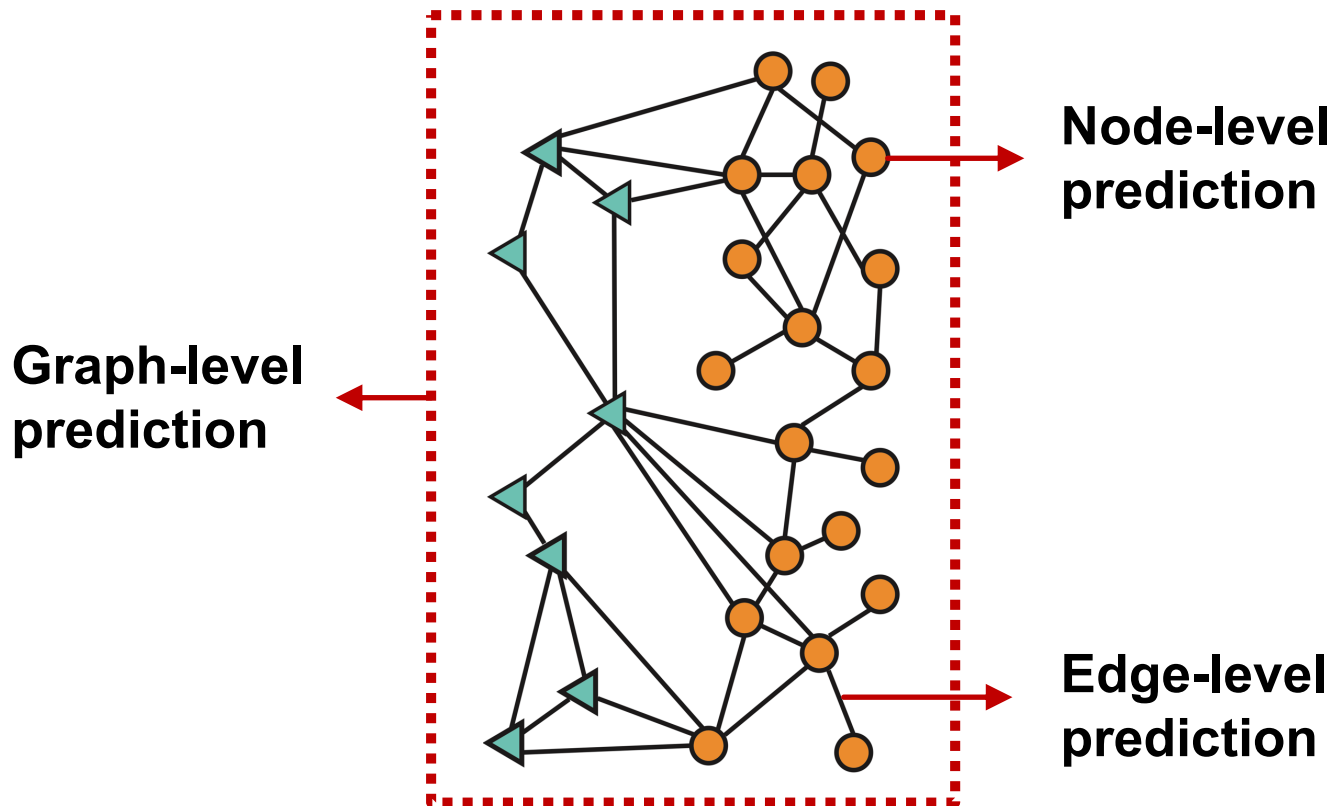# Prediction Heads: Node-level

- **Node-level prediction**: We can directly make prediction using node embeddings!
- 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
- $\widehat{\boldsymbol{y}}_v = \text{Head}_{\text{node}}(\mathbf{h}_v^{(L)}) = \mathbf{W}^{(H)}\mathbf{h}_v^{(L)}$
  - $\mathbf{W}^{(H)} \in \mathbb{R}^{k \times d}$: We map node embeddings from $\mathbf{h}_v^{(L)} \in \mathbb{R}^d$ to $\widehat{\boldsymbol{y}}_v \in \mathbb{R}^k$ so that we can compute the loss

Prediction Heads: Edge-level
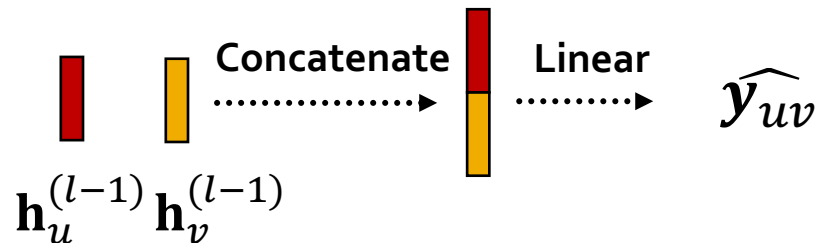
- **Edge-level prediction**: Make prediction using pairs of node embeddings
- Suppose we want to make k-way prediction
- $\widehat{\boldsymbol{y}}_{\boldsymbol{uv}} = \text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$
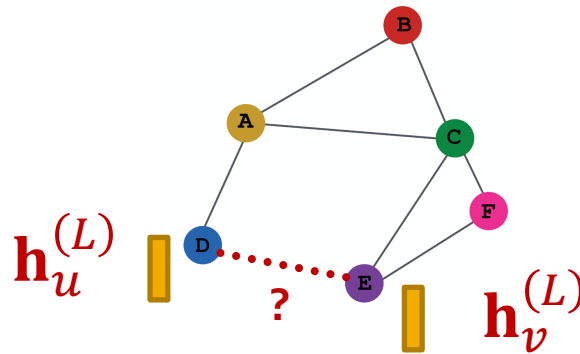
- What are the options for $\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$?

Prediction Heads: Edge-level

- **Options for** $\mathrm{Head}_{\mathrm{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$:
- **(1) Concatenation + Linear**

  - We have seen this in graph attention

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

Prediction Heads: Edge-level

- **Options for** $\text{Head}_{\text{edg}e}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$:

- **(2) Dot product**

  - $\widehat{\boldsymbol{y}}_{\boldsymbol{uv}} = (\mathbf{h}_u^{(L)})^T \mathbf{h}_v^{(L)}$

  - **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**: $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(k)}$ trainable

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

      $$\dots$$

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

      $$\widehat{\boldsymbol{y}}_{uv} = \text{Concat}(\widehat{\boldsymbol{y}}_{\boldsymbol{uv}}^{\boldsymbol{(1)}}, \dots, \widehat{\boldsymbol{y}}_{\boldsymbol{uv}}^{\boldsymbol{(k)}}) \in \mathbb{R}^k$$

Prediction Heads: Graph-level

- **Graph-level prediction**: Make prediction using all the node embeddings in our graph
- Suppose we want to make k-way prediction
- $\widehat{\boldsymbol{y}}_G = \text{Head}_{\text{graph}}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$

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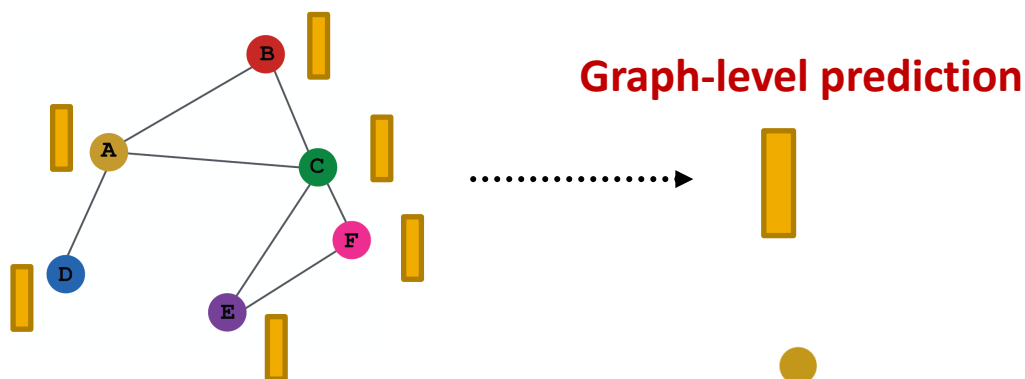
- $\text{Head}_{\text{graph}}(\cdot)$ is similar to $\text{AGG}(\cdot)$ in a GNN layer!

(2) Aggregation

(1) Message
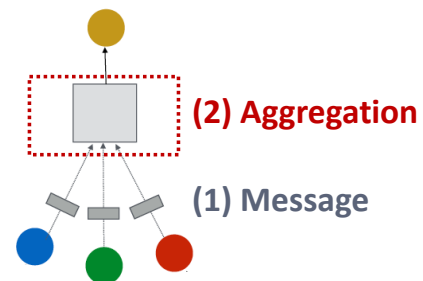
Prediction Heads: Graph-level

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

- **(1) Global mean pooling**

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

- **(2) Global max pooling**

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

- **(3) Global sum pooling**

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

- These options work great for small graphs

- **Can we do better for large graphs?**

Stanford CS224W: Training Graph Neural Networks

CS224W: Machine Learning with Graphs
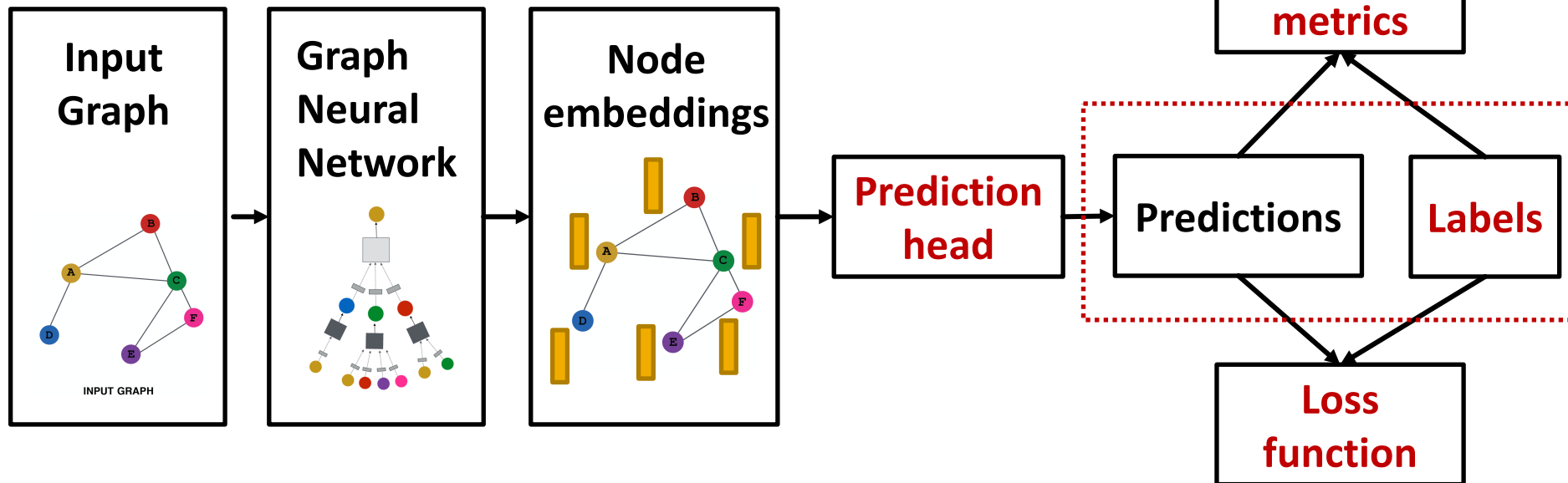Jure Leskovec, Stanford University
http://cs224w.stanford.edu

(2) Where does ground-truth come from?
- **Supervised labels**
- **Unsupervised signals**

Input Graph	Graph Neural Network	Node embeddings

→ **Prediction head** → **Predictions**

Labels

Evaluation metrics

Loss function

Supervised vs Unsupervised

- **Supervised learning on graphs**
  - **Labels come from external sources**
    - E.g., predict drug likeness of a molecular graph
- **Unsupervised learning on graphs**
  - **Signals come from graphs themselves**
    - E.g., link prediction: predict if two nodes are connected
- **Sometimes the differences are blurry**
  - We still have "supervision" in unsupervised learning
    - E.g., train a GNN to predict node clustering coefficient
 - An alternative name for "**unsupervised**" is "**self-supervised**"

Supervised Labels on Graphs

- **Supervised labels come from the specific use cases**. For example:

  - **Node labels y_v:** in a citation network, which subject area does a node belong to

  - **Edge labels y_{uv}:** in a transaction network, whether an edge is fraudulent

  - **Graph labels y_G:** among molecular graphs, the drug likeness of graphs

- **Advice:** Reduce your task to node / edge / graph labels, since they are easy to work with

  - **E.g.,** we knew some nodes form a cluster. We can treat the cluster that a node belongs to as a **node label**

Unsupervised Signals on Graphs

- **The problem:** sometimes **we only have a graph, without any external labels**
- **The solution:** "self-supervised learning", we can find supervision signals within the graph.
  - For example, we can **let GNN predict the following:**
  - **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, predict if two graphs are isomorphic
  - **These tasks do not require any external labels!**

GNN Training Pipeline (3)
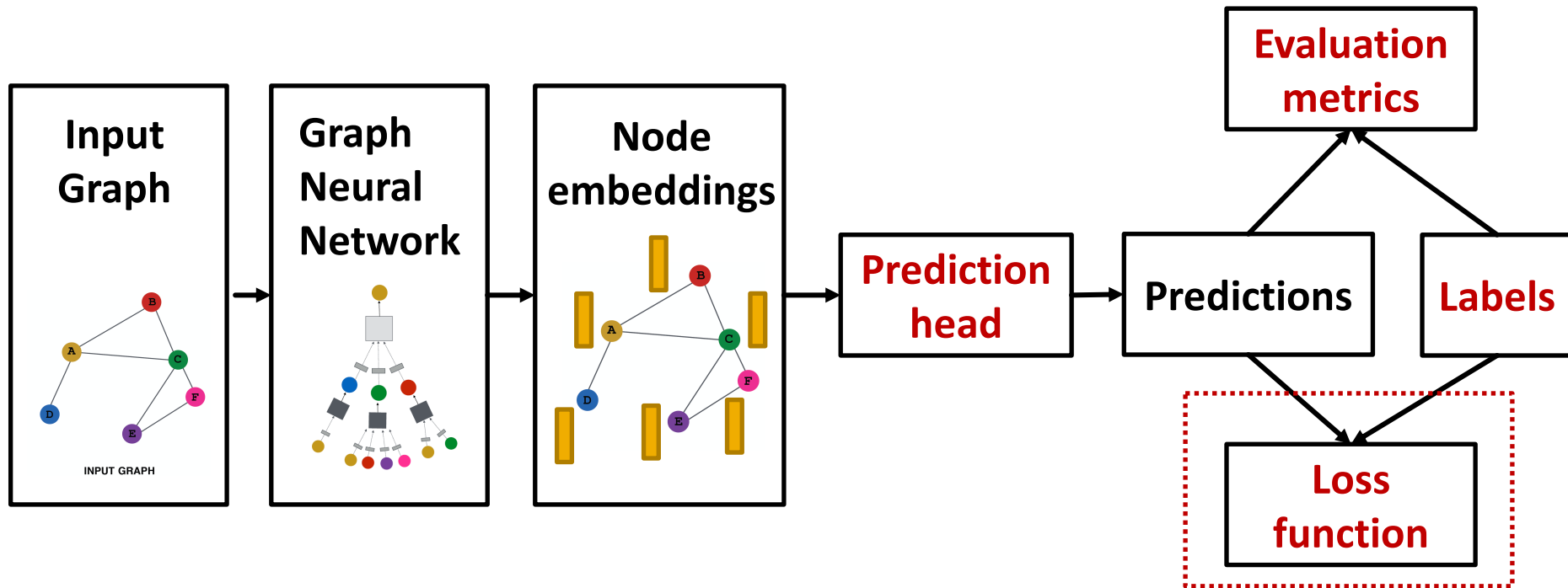
(3) How do we compute the final loss?
- **Classification loss**
- **Regression loss**

Settings for GNN Training

- **The setting:** We have N data points

  - Each data point can be a node/edge/graph

  - **Node-level**: prediction $\widehat{\boldsymbol{y}}_v^{(i)}$, label $\boldsymbol{y}_v^{(i)}$

  - **Edge-level**: prediction $\widehat{\boldsymbol{y}}_{uv}^{(i)}$, label $\boldsymbol{y}_{uv}^{(i)}$

  - **Graph-level**: prediction $\widehat{\boldsymbol{y}}_G^{(i)}$, label $\boldsymbol{y}_G^{(i)}$

  - We will use prediction $\widehat{\boldsymbol{y}}^{(i)}$, label $\boldsymbol{y}^{(i)}$ to refer **predictions at all levels**

Classification or Regression

- **Classification**: labels $y^{(i)}$ with discrete value
  - E.g., Node classification: which category does a node belong to
- **Regression**: labels $y^{(i)}$ with continuous value
  - E.g., predict the drug likeness of a molecular graph
- GNNs can be applied to both settings
- **Differences: loss function & evaluation metrics**

Classification Loss

- As discussed in lecture 6, **cross entropy (CE)** is a very common loss function in classification

- K-way prediction for i-th data point:

$$\text{CE}\left(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}\right) = -\sum_{j=1}^{K} \boldsymbol{y}_j^{(i)} \log\left(\widehat{\boldsymbol{y}}_j^{(i)}\right)$$

Label   Prediction   *i*-th data point   *j*-th class

where:

E.g.

0	0	1	0	0

$$\boldsymbol{y}^{(i)} \in \mathbb{R}^K = \text{one-hot label encoding}$$
$$\widehat{\boldsymbol{y}}^{(i)} \in \mathbb{R}^K = \text{prediction after Softmax}(\cdot)$$

E.g.

0.1	0.3	0.4	0.1	0.1

- Total loss over all N training examples

$$\text{Loss} = \sum_{i=1}^{N} \text{CE}\left(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}\right)$$

Regression Loss

- For regression tasks we often use **Mean Squared Error (MSE)** a.k.a. **L2 loss**
- K-way regression for data point (i):

$$\text{MSE}\big(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}\big) = \sum_{j=1}^{K} \big(\boldsymbol{y}_j^{(i)} - \widehat{\boldsymbol{y}}_j^{(i)}\big)^2$$

i-th data point

j-th target

where:

E.g.	1.4	2.3	1.0	0.5	0.6

$\boldsymbol{y}^{(i)} \in \mathbb{R}^k$ = Real valued vector of targets
$\widehat{\boldsymbol{y}}^{(i)} \in \mathbb{R}^k$ = Real valued vector of predictions

E.g.	0.9	2.8	2.0	0.3	0.8

- Total loss over all N training examples

$$\text{Loss} = \sum_{i=1}^{N} \text{MSE}\big(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}\big)$$

GNN Training Pipeline (4)

(4) How do we measure the success of a GNN?
- **Accuracy**
- **ROC AUC**

Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu
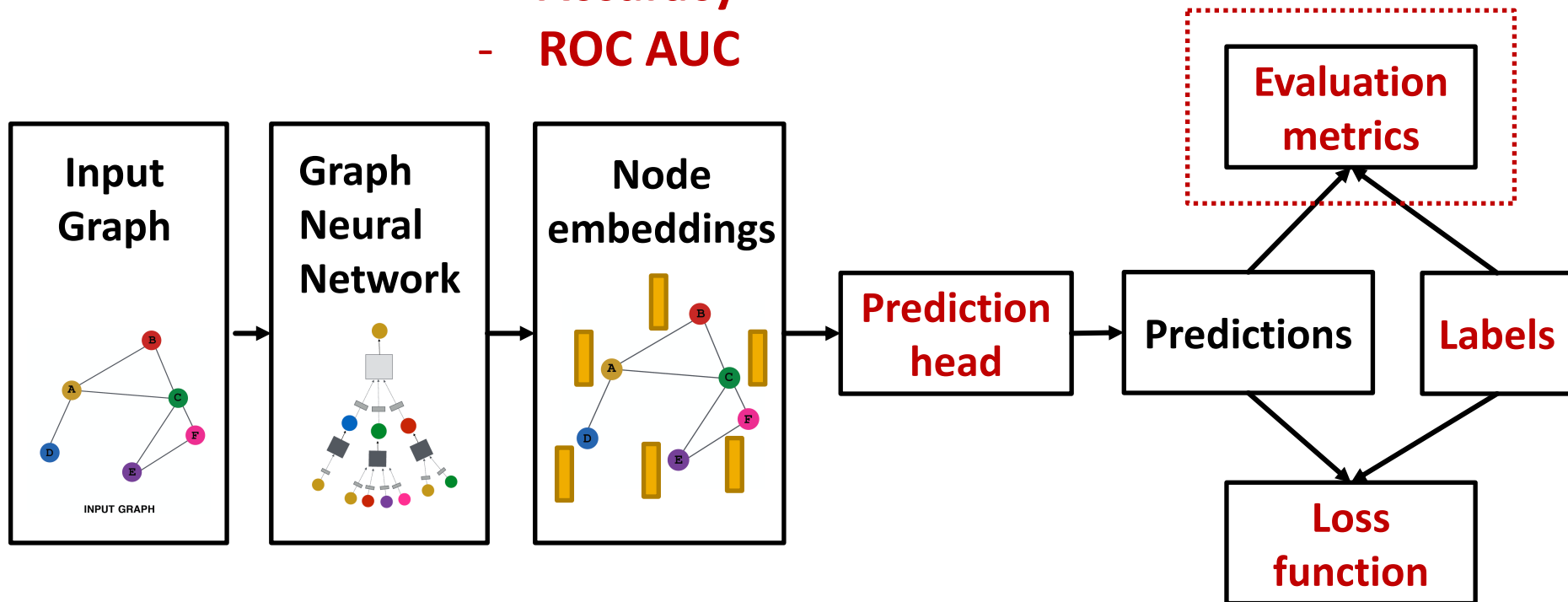
- **We use standard evaluation metrics for GNN**
  - (Content below can be found in any ML course)
  - In practice we will use [sklearn](#) for implementation
  - Suppose we make predictions for N data points
- **Evaluate regression tasks on graphs:**

- **Root mean square error (RMSE)**

$$\sqrt{\sum_{i=1}^{N} \frac{(y^{(i)} - \hat{y}^{(i)})^2}{N}}$$

- **Mean absolute error (MAE)**

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

Evaluation Metrics: Classification

- **Evaluate classification tasks on graphs:**
- **(1) Multi-class classification**
  - **We simply report the accuracy**

$$\frac{1\left[\mathrm{argmax}\left(\widehat{\boldsymbol{y}}^{(i)}\right) = \boldsymbol{y}^{(i)}\right]}{N}$$

- **(2) Binary classification**
  - Metrics sensitive to classification threshold
    - **Accuracy**
    - **Precision / Recall**
    - If the range of prediction is $[0,1]$, we will use 0.5 as threshold
  - Metric Agnostic to classification threshold
    - **ROC AUC**

Metrics for Binary Classification

- **Accuracy**:

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

- **Precision (P)**:

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

- **Recall (R)**:

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

- **F1-Score**:

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

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)

Sklearn Classification Report

- **ROC Curve:** Captures the tradeoff in TPR and FPR **as the classification threshold is varied for a binary classifier.**

TPR

FPR

NetChop C-term 3.0
TAP + ProteaSMM-i
ProteaSMM-i
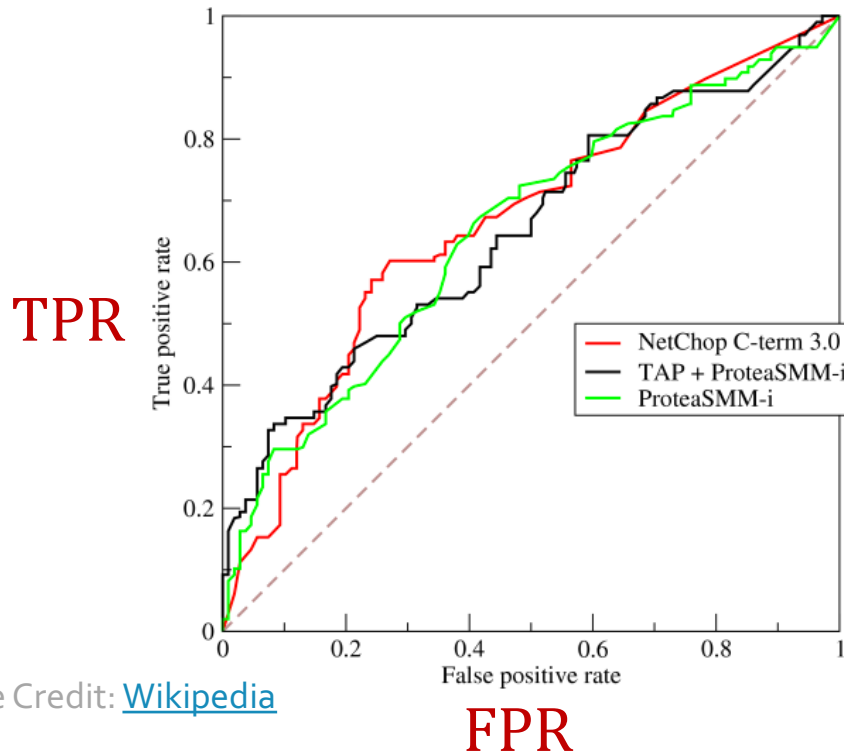
Image Credit: Wikipedia

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

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

Note: the dashed line represents **performance of a random classifier**

(4) Evaluation Metrics
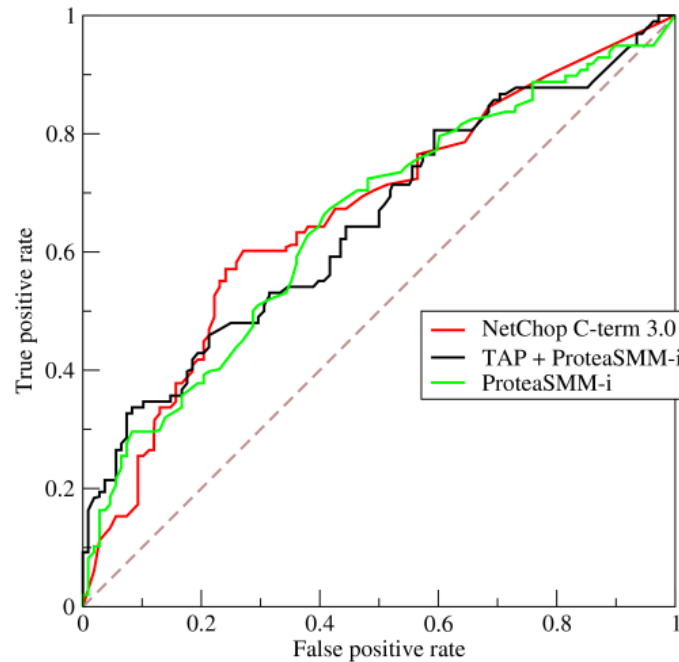
Content Credit: Wikipedia

- **ROC AUC:** **Area under the ROC Curve**.
- **Intuition:** The probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one

Stanford CS224W: Setting-up GNN Prediction Tasks

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu
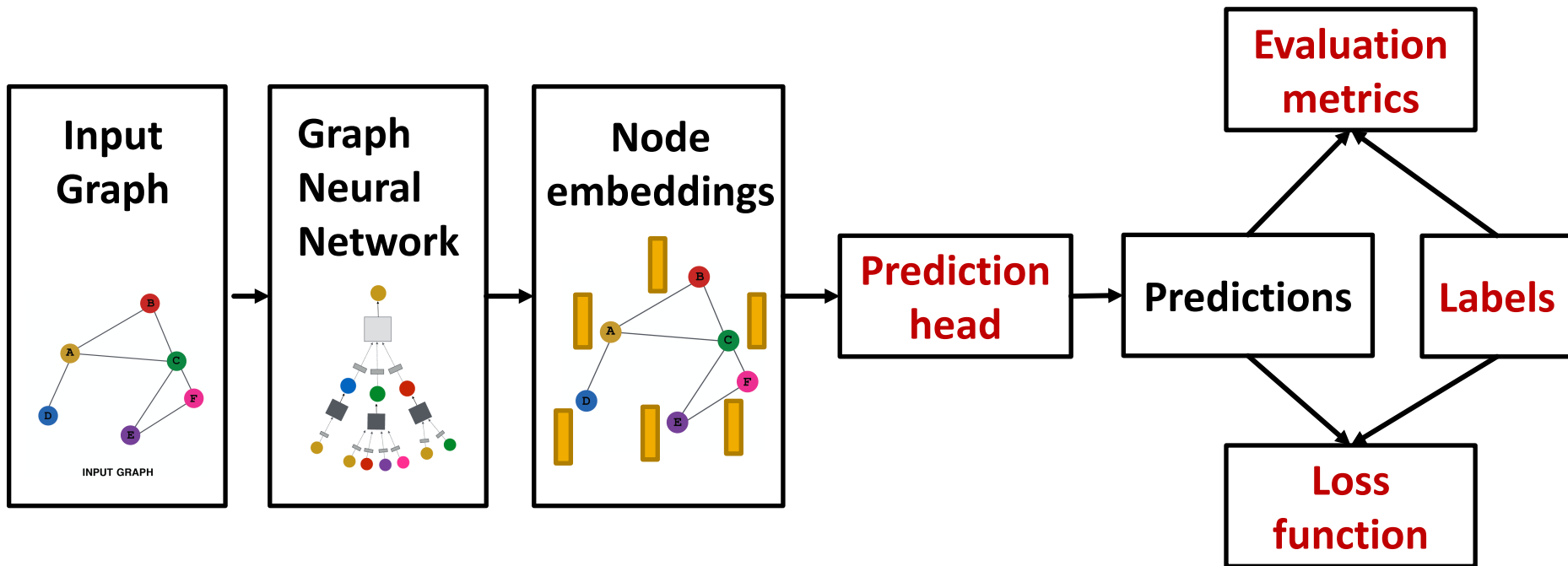
(5) How do we split our dataset into train / validation / test set?
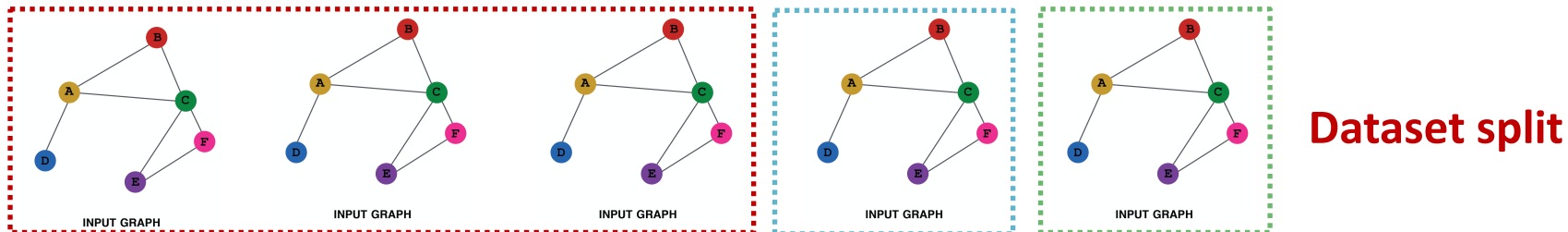
Dataset split

Dataset Split: Fixed / Random Split

- **Fixed split:** We will split our dataset **once**
  - **Training set**: used for optimizing GNN parameters
  - **Validation set**: develop model/hyperparameters
  - **Test set**: held out until we report final performance
- **A concern:** sometimes we cannot guarantee that the test set will really be held out
- **Random split:** we will **randomly split** our dataset into training / validation / test
  - We report **average performance over different random seeds**

Why Splitting Graphs is Special

- **Suppose we want to split an image dataset**

  - **Image classification: Each data point is an image**

  - Here **data points are independent**

    - **Image 5 will not affect our prediction on image 1**

Training

Validation

Test
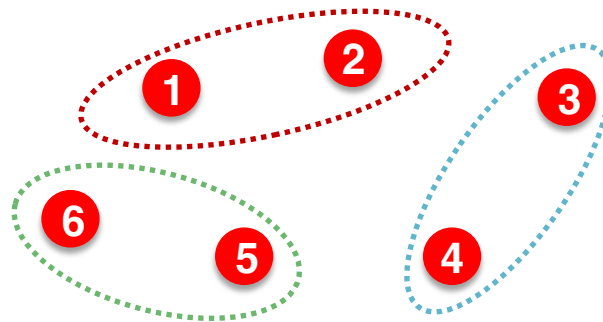
Why Splitting Graphs is Special

- **Splitting a graph dataset is different!**
    - **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
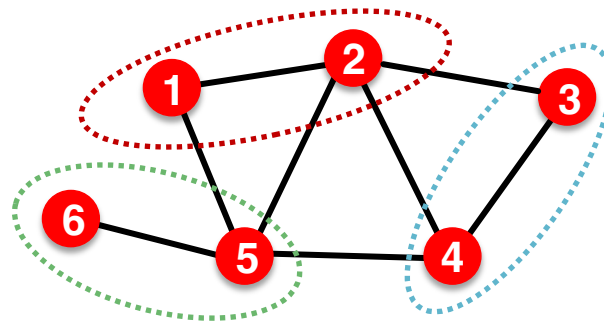
Test

- **What are our options?**

Why Splitting Graphs is Special

- **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**

Training

Validation

Test
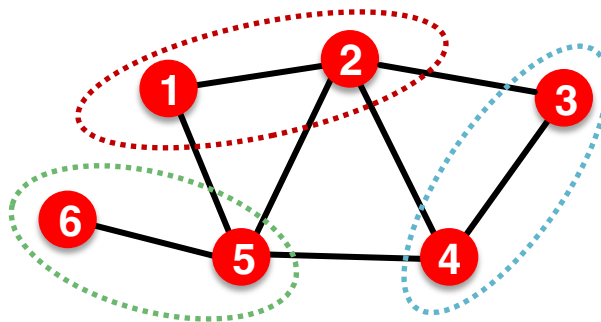
Why Splitting Graphs is Special

- **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
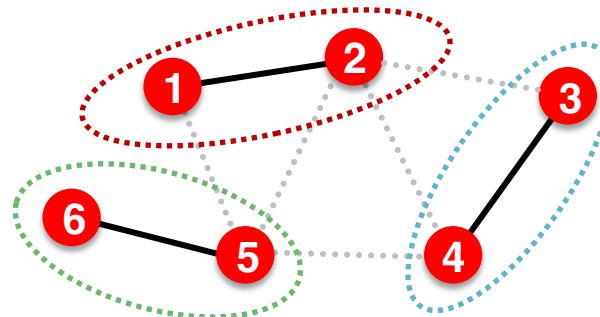
Validation

Test

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

Example: Node Classification

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


Training

Validation
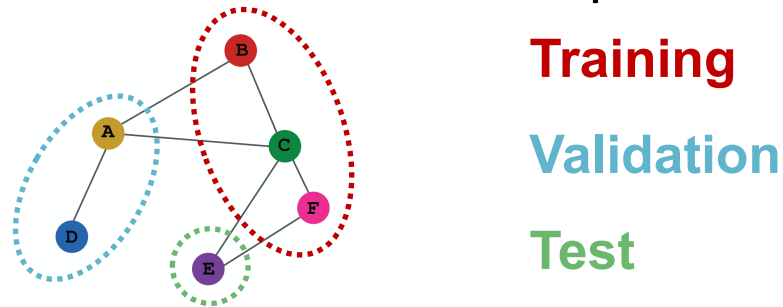
Test

- **Inductive** node classification
  - Suppose we have a dataset of 3 graphs
 - **Each split contains an independent graph**

Training

Validation

Test

Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu
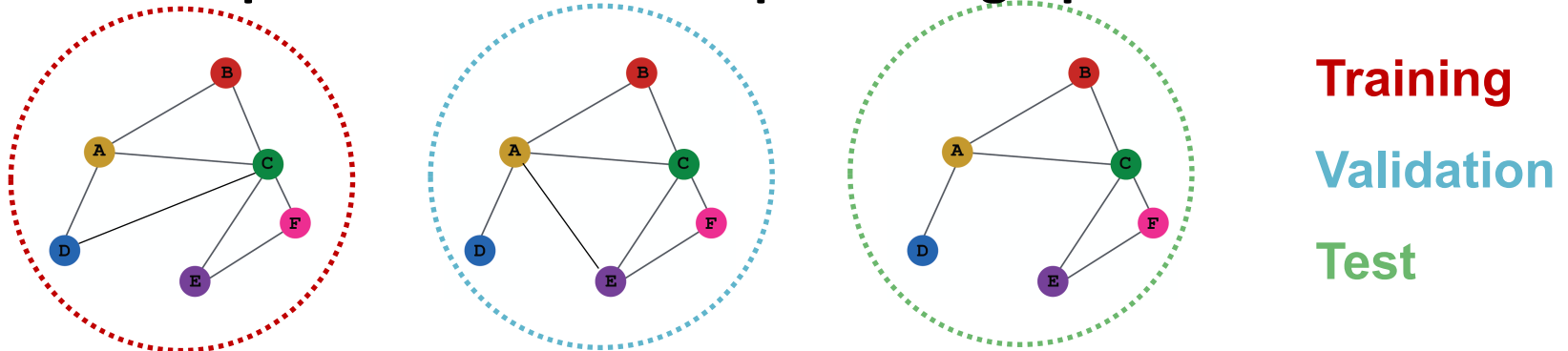
Example: Graph Classification

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

  - Because **we have to test on unseen graphs**

  - Suppose we have a dataset of 5 graphs. Each split will contain independent graph(s).
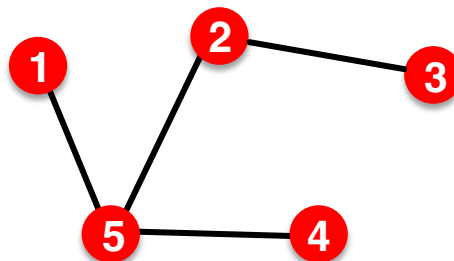

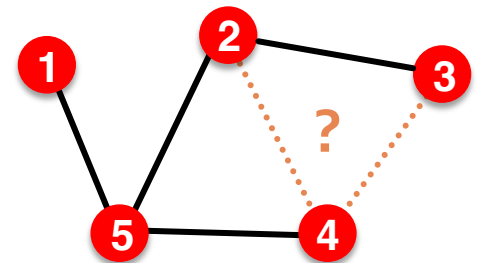**Training**   **Validation**   **Test**
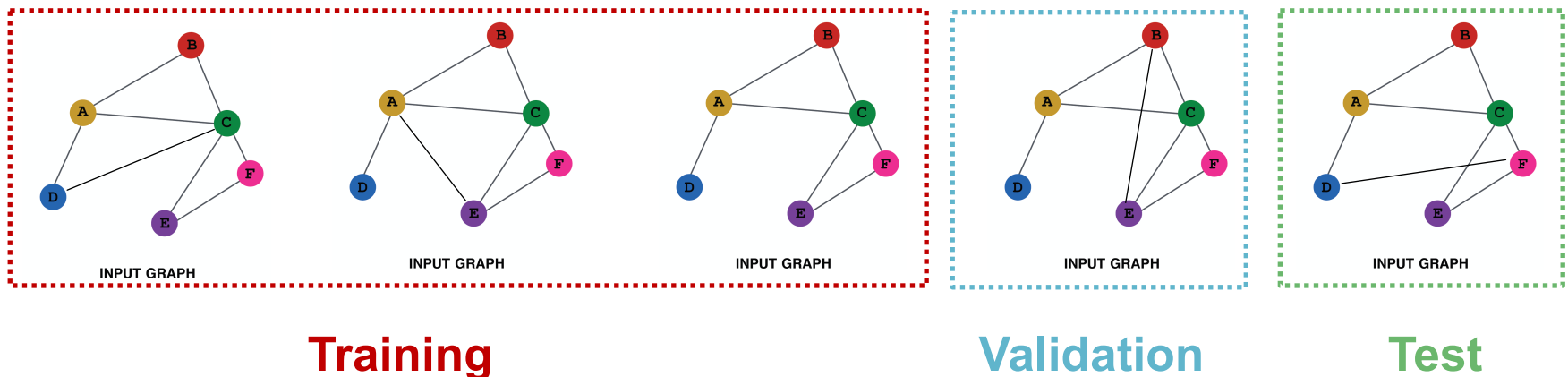
Example: Link Prediction

- **Goal of link prediction**: **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
  - Concretely, we need to **hide some edges from the GNN** and the **let the GNN predict if the edges exist**
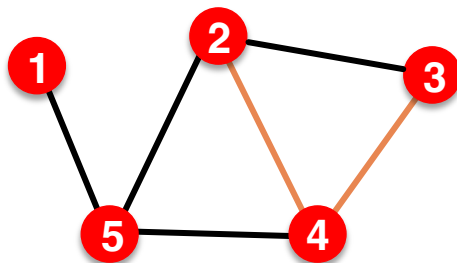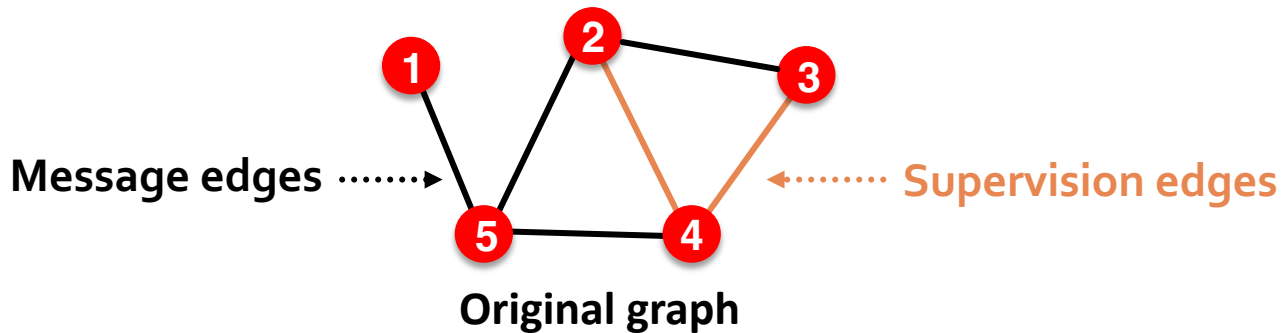

**Original graph**   **Input graph to GNN**   **Predictions made by GNN**

Setting up Link Prediction

Original graph

- **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!**

Setting up Link Prediction

- **Step 2: Split edges into train / validation / test**
- **Option 1: Inductive link prediction split**
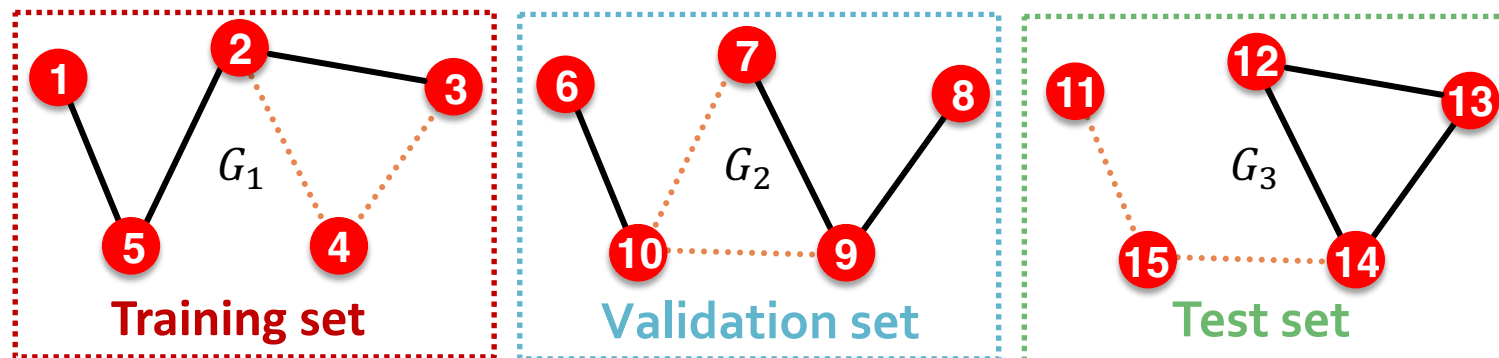 - **Suppose we have a dataset of 3 graphs. Each inductive split will contain an independent graph**

Training set   Validation set   Test set
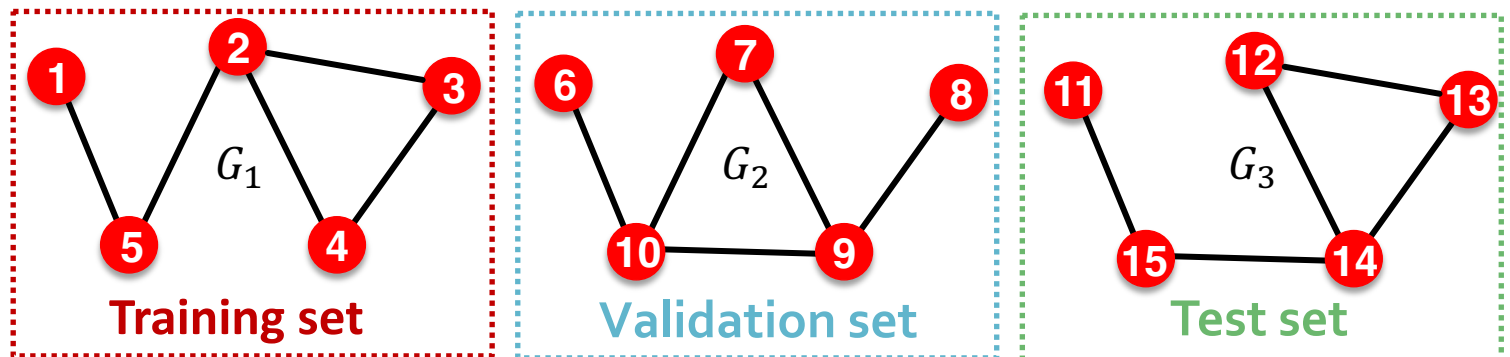
Setting up Link Prediction

- **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**
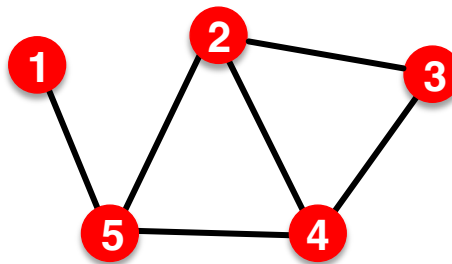   - **Supervision edges** are not the input to GNN

Message edge ———

Supervision edge ·········

G_1 Training set G_2 Validation set G_3 Test set

Setting up Link Prediction

- **Option 2: Transductive link prediction split:**
    - **This is the default setting when people talk about link prediction**
    - **Suppose we have a dataset of 1 graph**

Setting up Link Prediction

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

  - **By definition of "transductive", the entire graph can be observed in all dataset splits**

    - **But since edges are both part of graph structure and the supervision, we need to hold out validation / test edges**

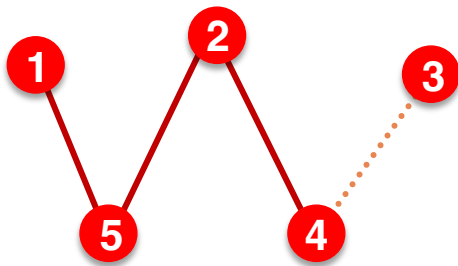   - **To train the training set, we further need to hold out supervision edges for the training set**
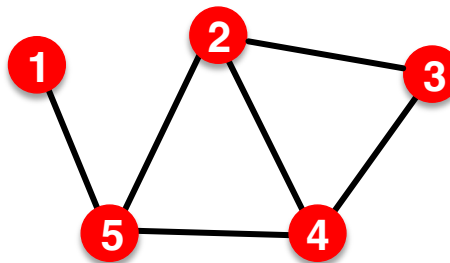
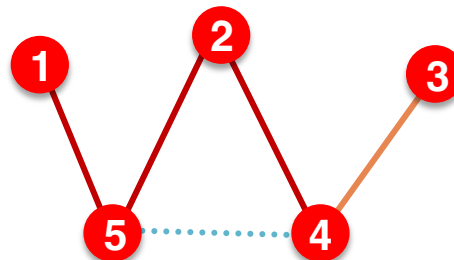 - **Next:** we will show the exact settings

Setting up Link Prediction

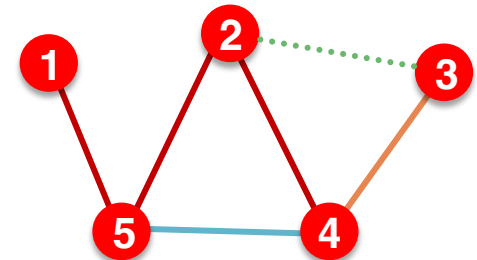- **Option 2: Transductive link prediction split:**
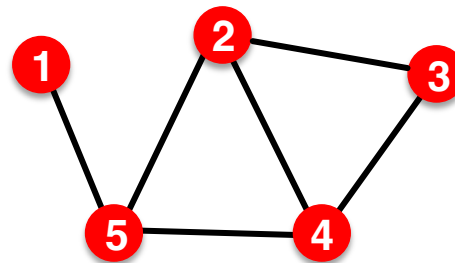
The original graph

(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**
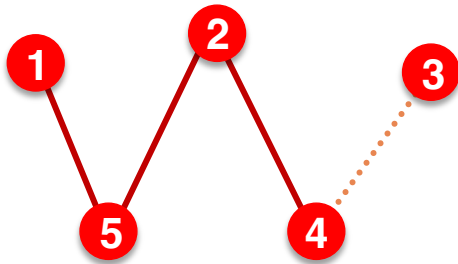
Setting up Link Prediction

- ## Option 2: Transductive link prediction split:
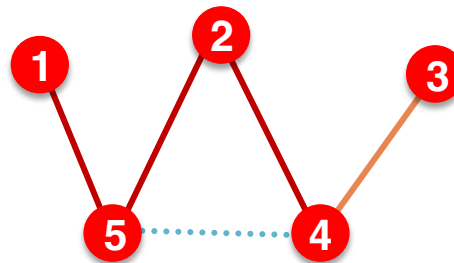
Why do we use growing number of edges?
After training, **supervision edges are known to GNN.**
Therefore, **an ideal model should use supervision edges in message passing** at validation time.
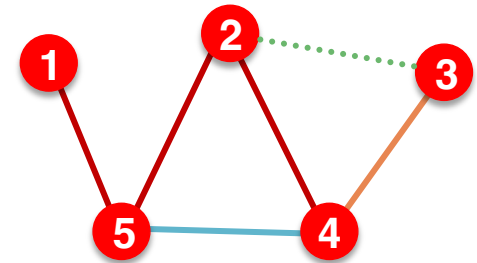The same applies to the test time.

(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**
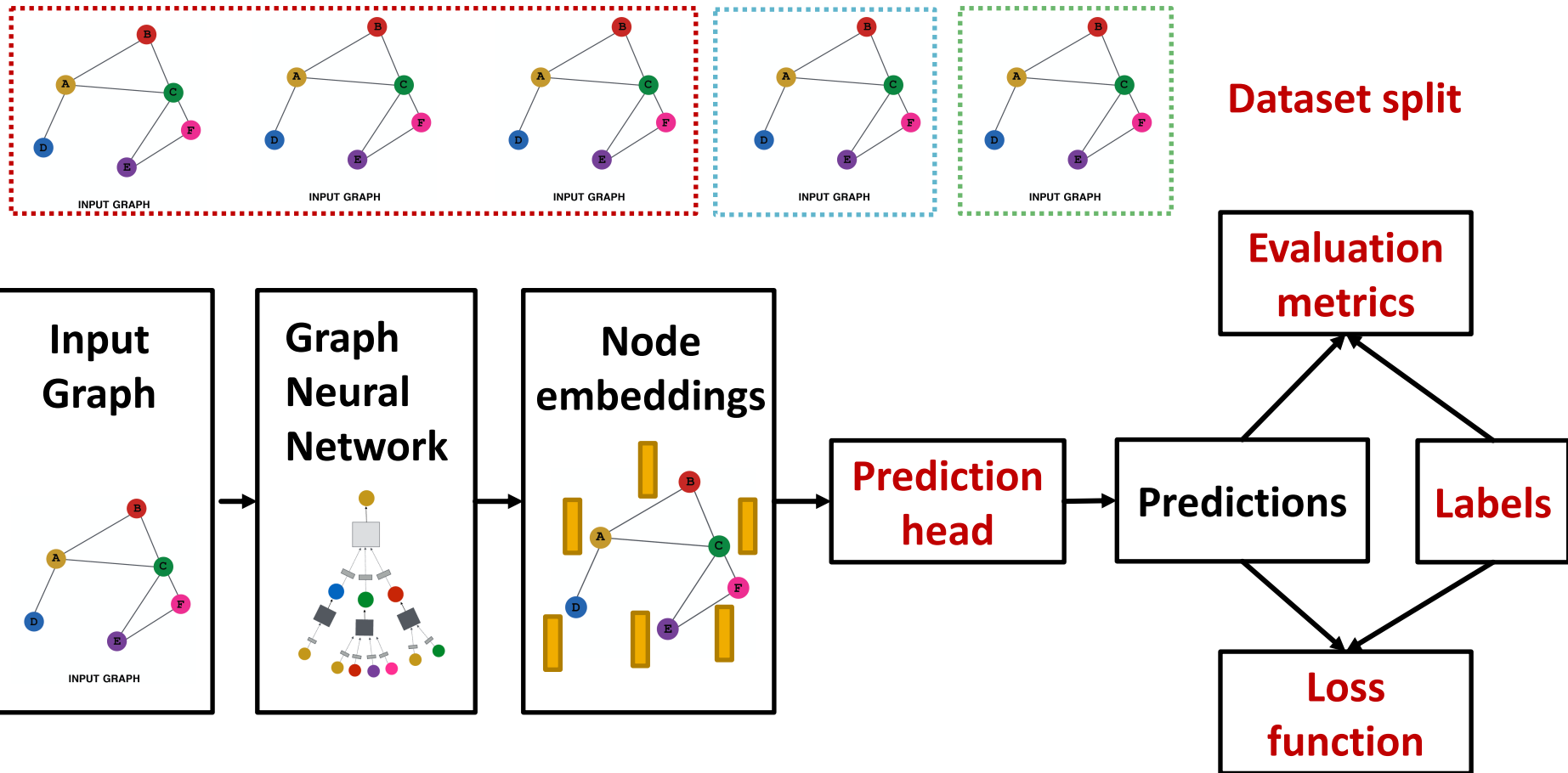
- ## Summary: Transductive link prediction split:

The original graph

Split

Split Graph with
4 types of edges

Training message edges
Training supervision edges
Validation edges
Test edges
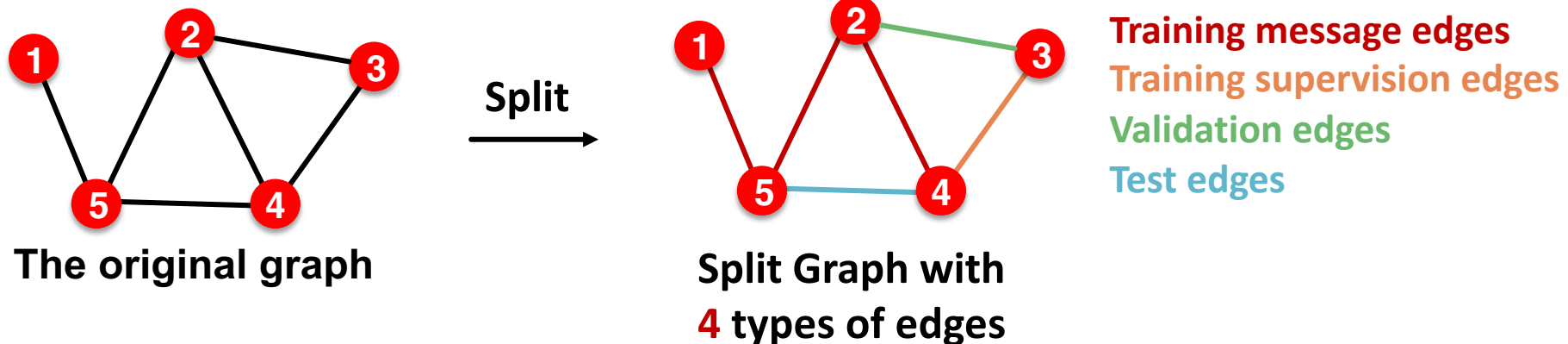
- **Note:** Link prediction settings are tricky and complex. You may find papers do link prediction differently.
- Luckily, we have full support in **PyG and GraphGym**

GNN Training Pipeline

Dataset split

Evaluation metrics

| Input Graph | → | Graph Neural Network | → | Node embeddings | → | **Prediction head** | → | Predictions |

Labels

Loss function

Implementation resources:

[GraphGym](#) further implements the full pipeline to facilitate GNN design

Summary of the Lecture

- **We introduce a general GNN framework:**
  - **GNN Layer**:
    - Transformation + Aggregation
    - Classic GNN layers: GCN, GraphSAGE, GAT
  - **Layer connectivity**:
    - The over-smoothing problem
    - Solution: skip connections
  - **Graph Augmentation:**
    - Feature augmentation
    - Structure augmentation
  - **Learning Objectives**
    - The full training pipeline of a GNN