Stanford CS224W: GNN Augmentation and Training

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

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 from our course last year as examples of this type of project

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.

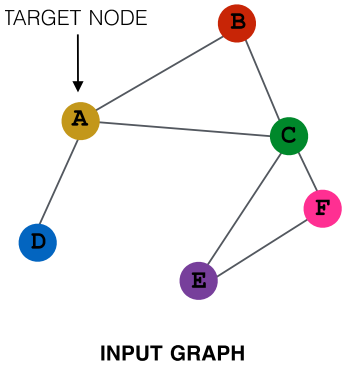
Project logistics

- Project is worth 20% of your course grade
  - Project proposal (2 pages), due February 7
  - Final reports, due March 21
- We recommend groups of 3, but groups of 2 are also allowed
- **Full project description will be released tonight**! We will provide much more detail on each project type, examples, pointers to datasets, tips for writing blog posts and Google Colabs, etc.

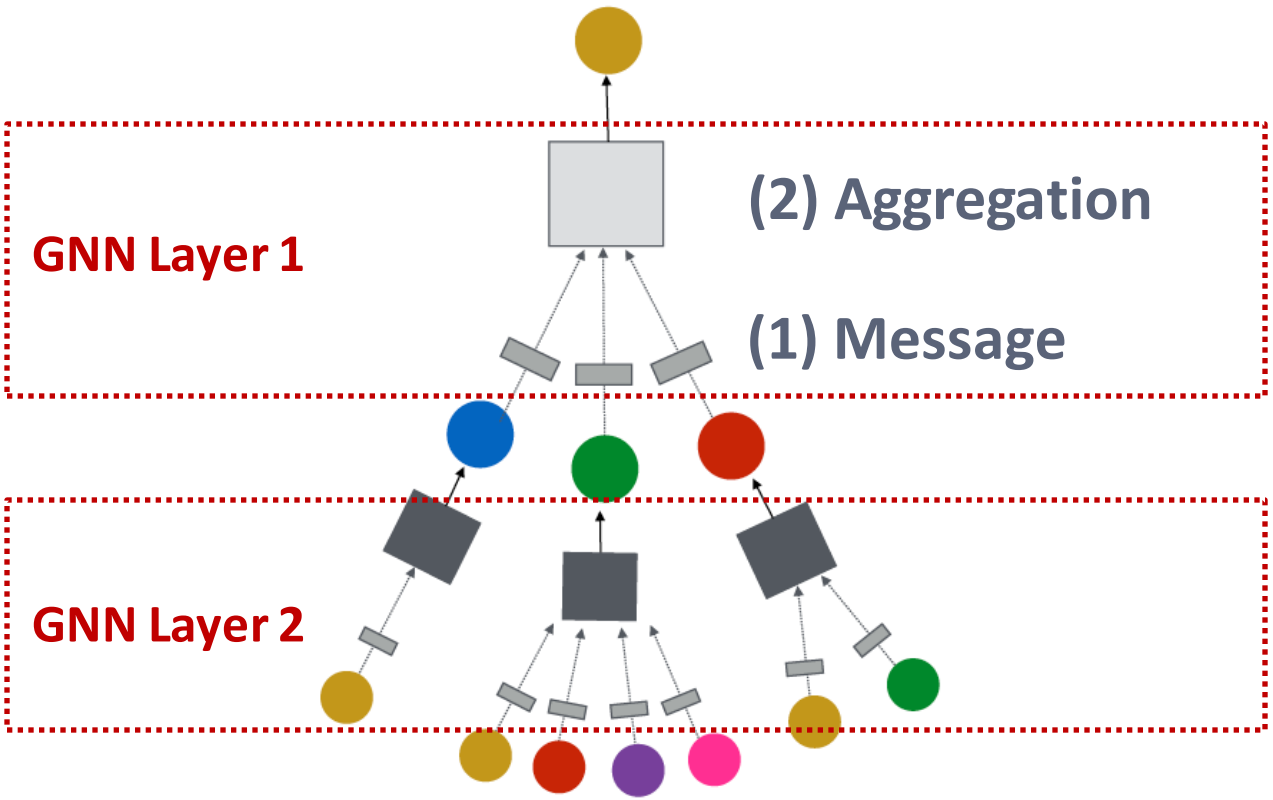
Stanford CS224W: GNN Augmentation and Training

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

Recap: A General GNN Framework



(5) Learning objective

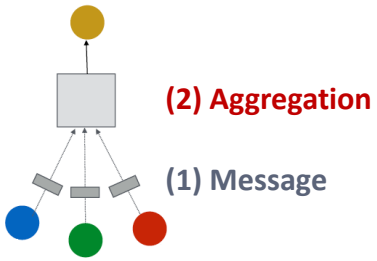


(3) Layer connectivity

(4) Graph augmentation

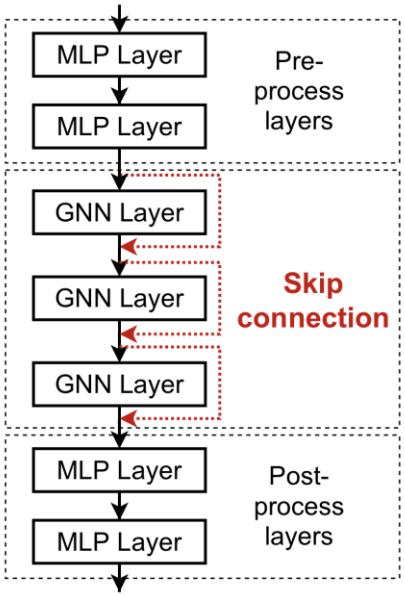
Recap: A Single GNN Layer

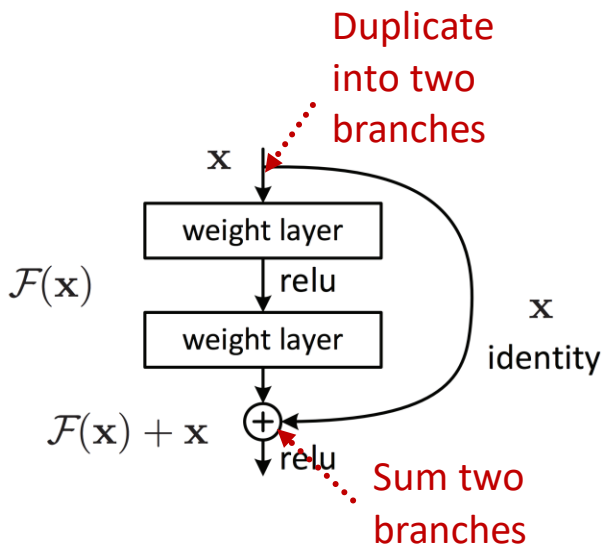
- **Putting things together:**
  - **(1) Message**: each node computes a message
$$\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), u \in \{N(v) \cup v\}$$
  - **(2) Aggregation**: aggregate messages from neighbors
$$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}, \mathbf{m}_v^{(l)}\right)$$
  - **Nonlinearity (activation):** Adds expressiveness
    - Often written as $\sigma(\cdot)$: $\mathrm{ReLU}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, ...
    - Can be added to **message or aggregation**



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:

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

After adding shortcuts:

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

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?

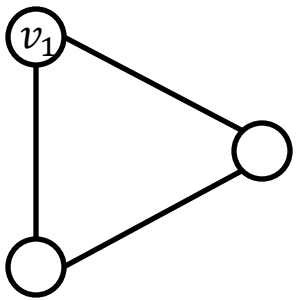
- **(2) Certain structures are hard to learn by GNN**
- **Solution:**
  - We can use cycle count as augmented node features

We start from cycle with length 0

Augmented node feature for v_1

[0, 0, 0, 1, 0, 0]

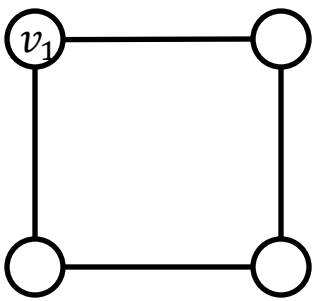
v_1 resides in a cycle with length 3



Augmented node feature for v_1

[0, 0, 0, 0, 1, 0]

v_1 resides in a cycle with length 4



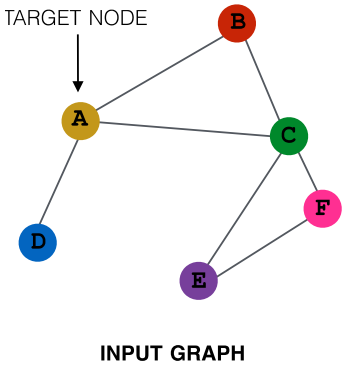
Stanford CS224W: Prediction with GNNs

CS224W: Machine Learning with Graphs

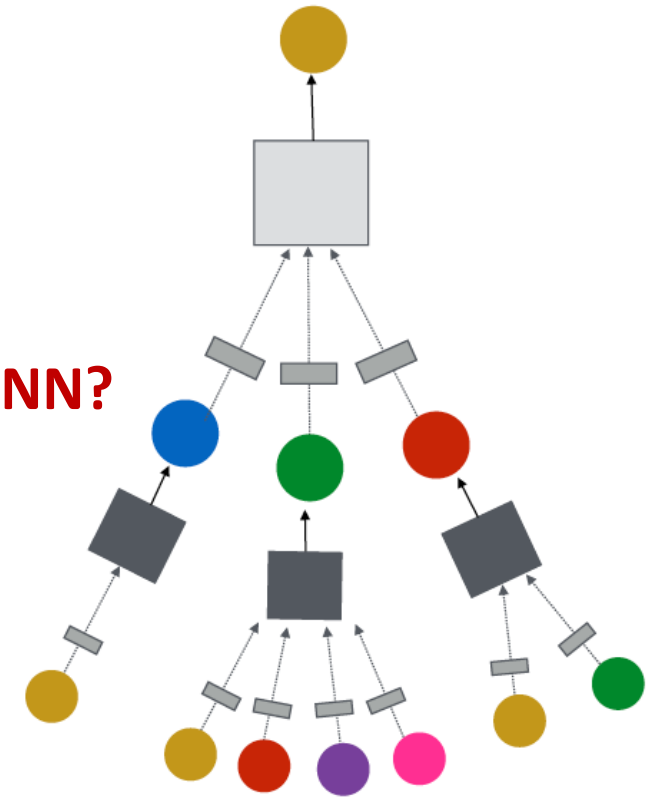
Jure Leskovec, Stanford University

http://cs224w.stanford.edu

A General GNN Framework (4)



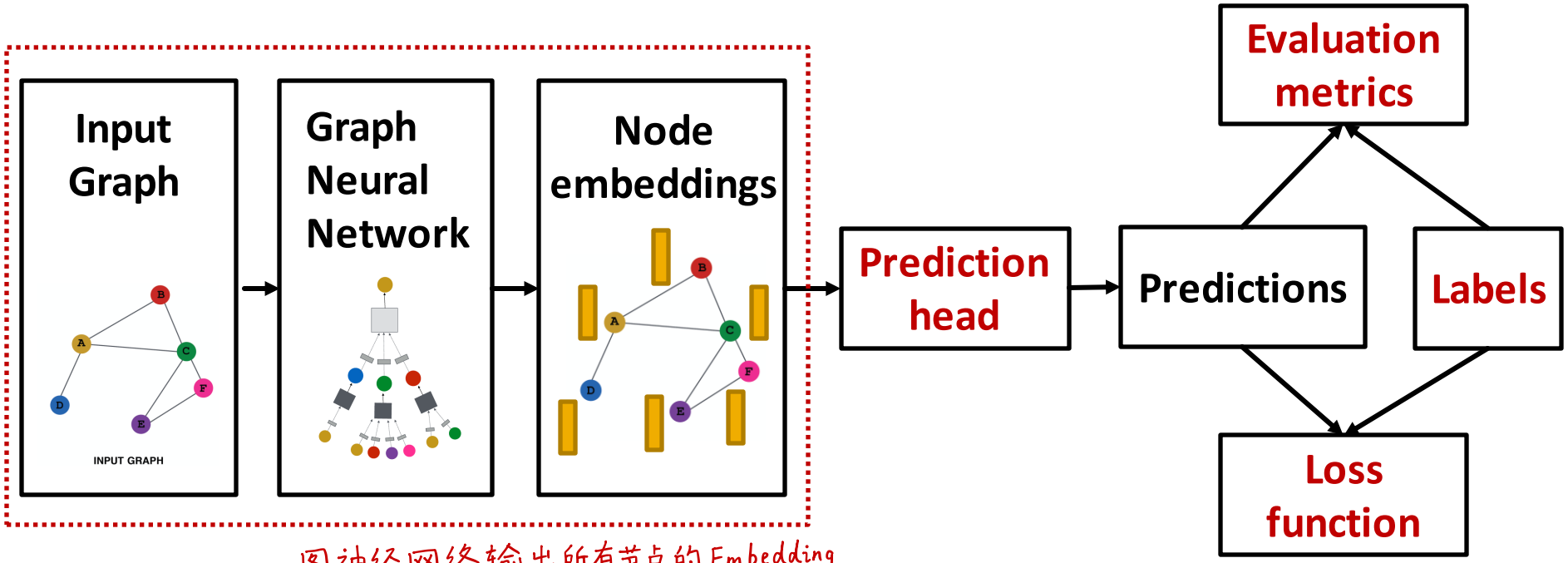
(5) Learning objective



Next: How do we train a GNN?

GNN Training Pipeline

So far what we have covered

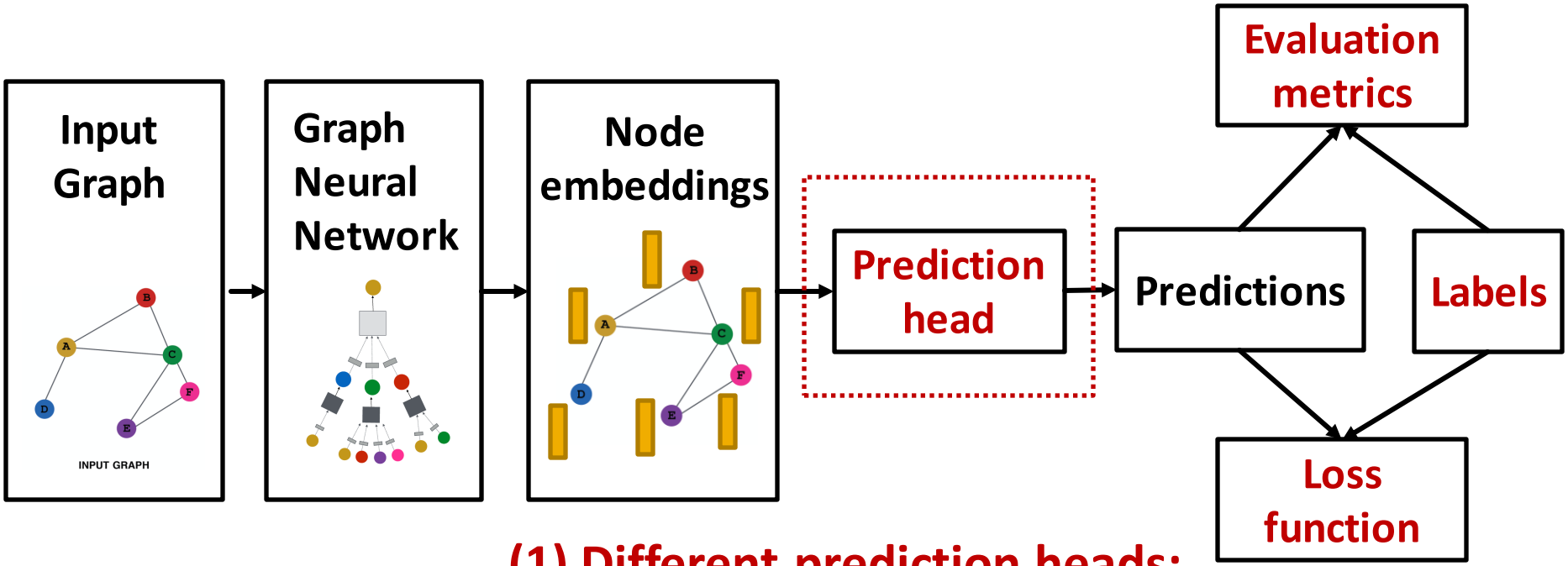


图神经网络输出所有节点的Embedding

Output of a GNN: set of node embeddings

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

GNN Training Pipeline (1)

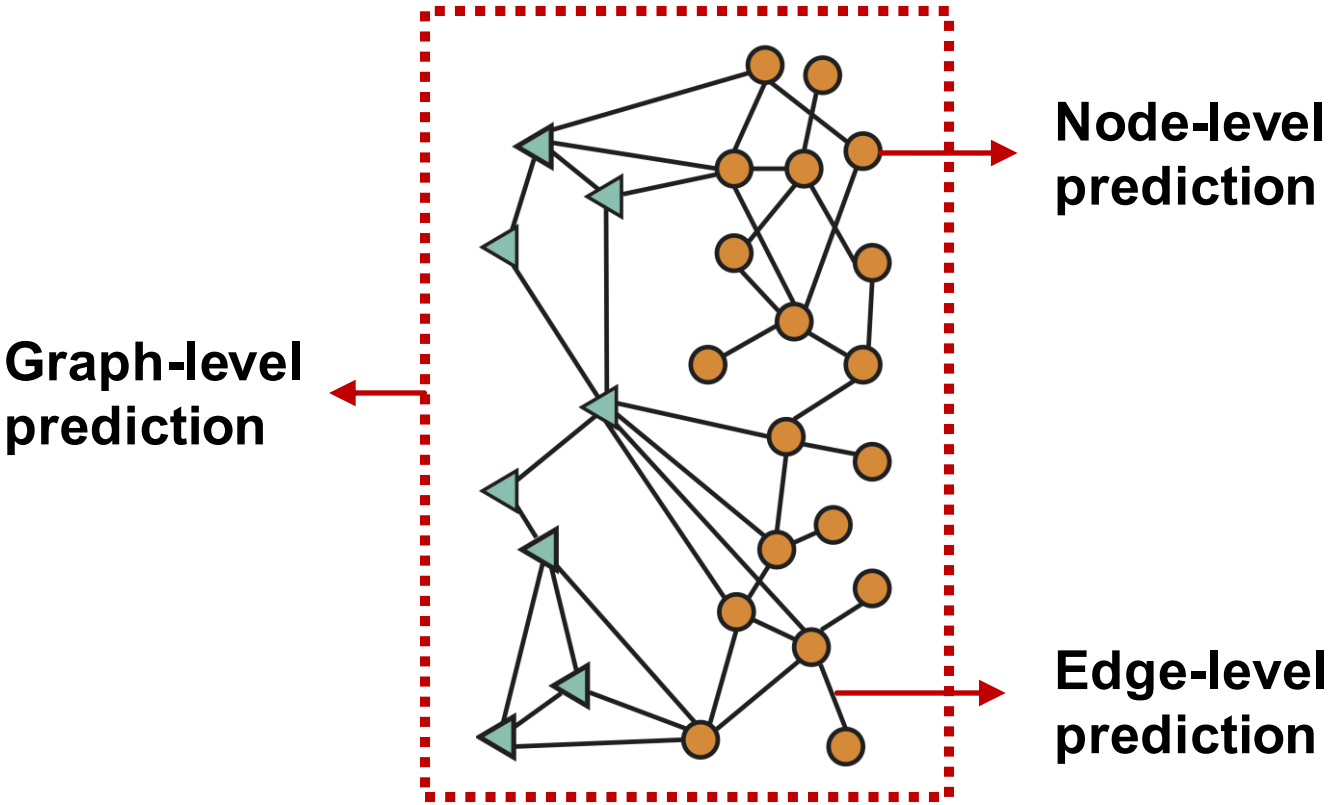


(1) Different prediction heads:

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

GNN Prediction Heads

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

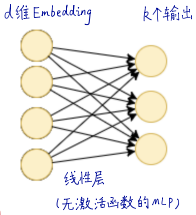


Prediction Heads: Node-level

- **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 分类:k个类别的概率
  - Regression: regress on k targets 回归：k个连续值

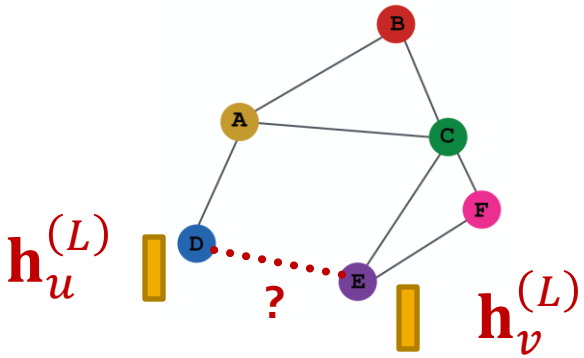
预测结果

- $\hat{\boldsymbol{y}}_v = \text{Head}_{\text{node}}(\mathbf{h}_v^{(L)}) = \mathbf{W}^{(H)}\mathbf{h}_v^{(L)}$ (head) 线性层
  - $\mathbf{W}^{(H)} \in \mathbb{R}^{k*d}$: 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



Prediction Heads: Edge-level

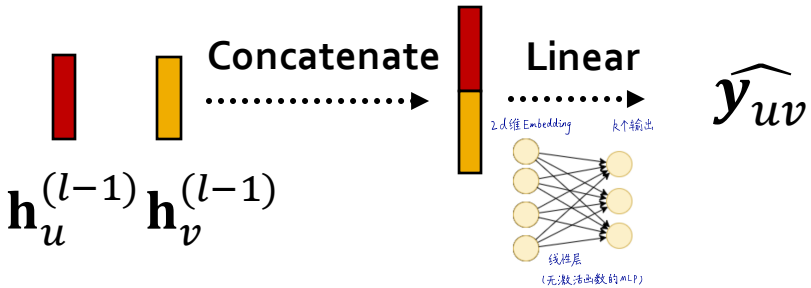
- **Edge-level prediction**: Make prediction using pairs of node embeddings
- Suppose we want to make k-way prediction
- $\hat{\boldsymbol{y}}_{\boldsymbol{uv}} = \text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$



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

Prediction Heads: Edge-level

- **Options for** $\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$:
- **(1) Concatenation + Linear**
  - We have seen this in graph attention



类似 GAT 中的 a 函数

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

映射

Prediction Heads: Edge-level

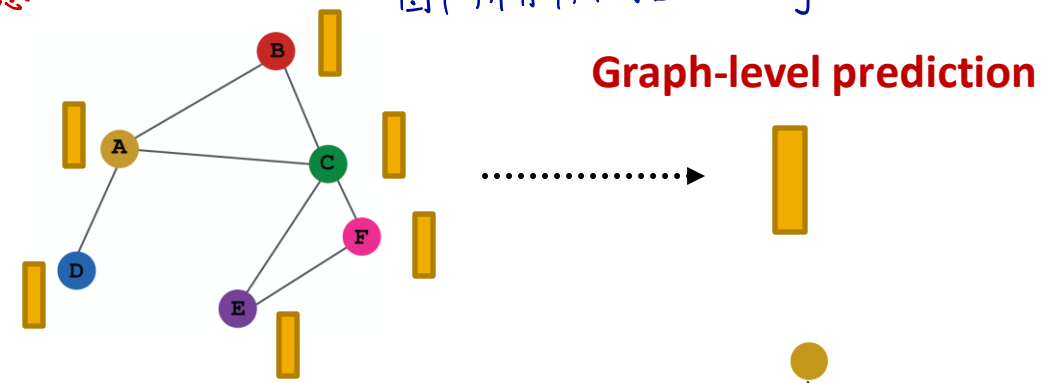
- **Options for** $\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$:
- **(2) Dot product**
  - $\hat{\boldsymbol{y}}_{uv} = (\mathbf{h}_u^{(L)})^T \mathbf{h}_v^{(L)}$
  - **This approach only applies to 1-way prediction** (e.g., link prediction: predict the existence of an edge)
  -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

$$\hat{\boldsymbol{y}}_{uv}^{(1)} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(1)} \mathbf{h}_v^{(L)}$$
$$\dots$$
$$\hat{\boldsymbol{y}}_{uv}^{(k)} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(k)} \mathbf{h}_v^{(L)}$$
$$\hat{\boldsymbol{y}}_{uv} = \text{Concat}(\hat{\boldsymbol{y}}_{uv}^{(1)}, \dots, \hat{\boldsymbol{y}}_{uv}^{(k)}) \in \mathbb{R}^k$$

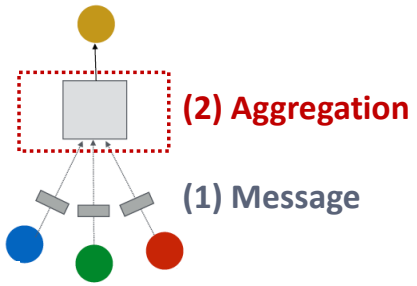
Prediction Heads: Graph-level

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

聚合信息. 图中 所有节点的 Embedding



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



Prediction Heads: Graph-level

全局池化：每个Embedding变成一个标量

- 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?**

Issue of Global Pooling

- **Issue:** Global pooling over a (large) graph will lose information
- **Toy example:** we use 1-dim node embeddings
  - Node embeddings for G_1: $\{-1, -2, 0, 1, 2\}$
  - Node embeddings for G_2: $\{-10, -20, 0, 10, 20\}$
  - Clearly G_1 and G_2 have very different node embeddings
    → Their structures should be different
- **If we do global sum pooling:**
  - **Prediction for** G_1: $\hat{y}_G = \text{Sum}(\{-1, -2, 0, 1, 2\}) = 0$
  - **Prediction for** G_2: $\hat{y}_G = \text{Sum}(\{-10, -20, 0, 10, 20\}) = 0$
  - We cannot differentiate G_1 and G_2! 求和：只看均值 不看方差

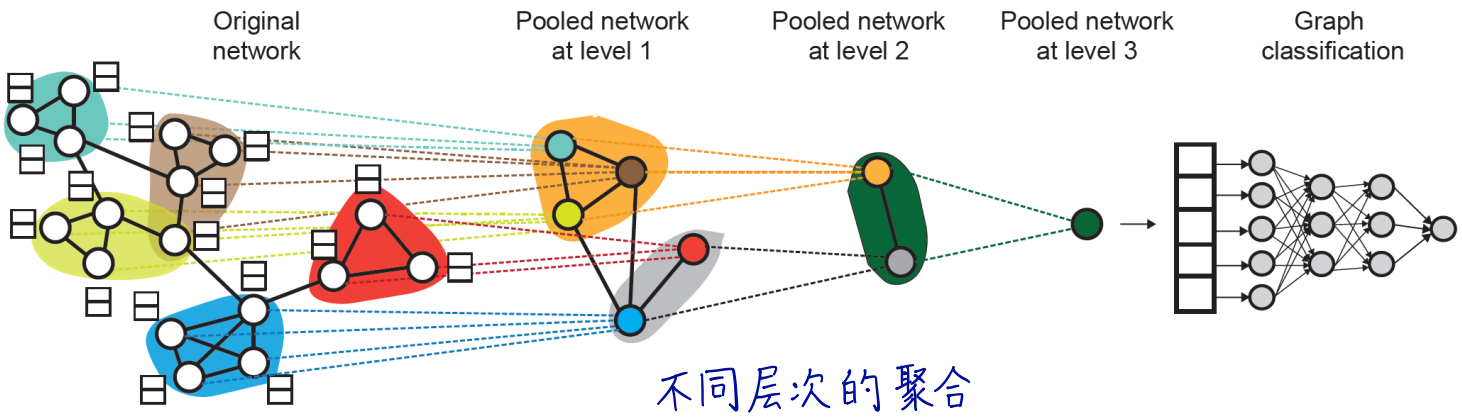
Hierarchical Global Pooling

- **A solution:** Let's aggregate all the node embeddings **hierarchically** 分层
  - **Toy example:** We will aggregate via $\mathrm{ReLU}(\mathrm{Sum}(\cdot))$
    - We first separately aggregate the first 2 nodes and last 3 nodes
    - Then we aggregate again to make the final prediction
  - G_1 node embeddings: $\{-1, -2, 0, 1, 2\}$
    - **Round 1**: $\hat{y}_a = \mathrm{ReLU}(\mathrm{Sum}(\{-1, -2\})) = 0$, $\hat{y}_b = \mathrm{ReLU}(\mathrm{Sum}(\{0, 1, 2\})) = 3$
    - **Round 2:** $\hat{y}_G = \mathrm{ReLU}(\mathrm{Sum}(\{y_a, y_b\})) = \mathbf{3}$
  - G_2 node embeddings: $\{-10, -20, 0, 10, 20\}$
    - **Round 1**: $\hat{y}_a = \mathrm{ReLU}(\mathrm{Sum}(\{-10, -20\})) = 0$, $\hat{y}_b = \mathrm{ReLU}(\mathrm{Sum}(\{0, 10, 20\})) = 30$
    - **Round 2:** $\hat{y}_G = \mathrm{ReLU}(\mathrm{Sum}(\{y_a, y_b\})) = \mathbf{30}$

Now we can differentiate G_1 and G_2 !

Hierarchical Pooling In Practice

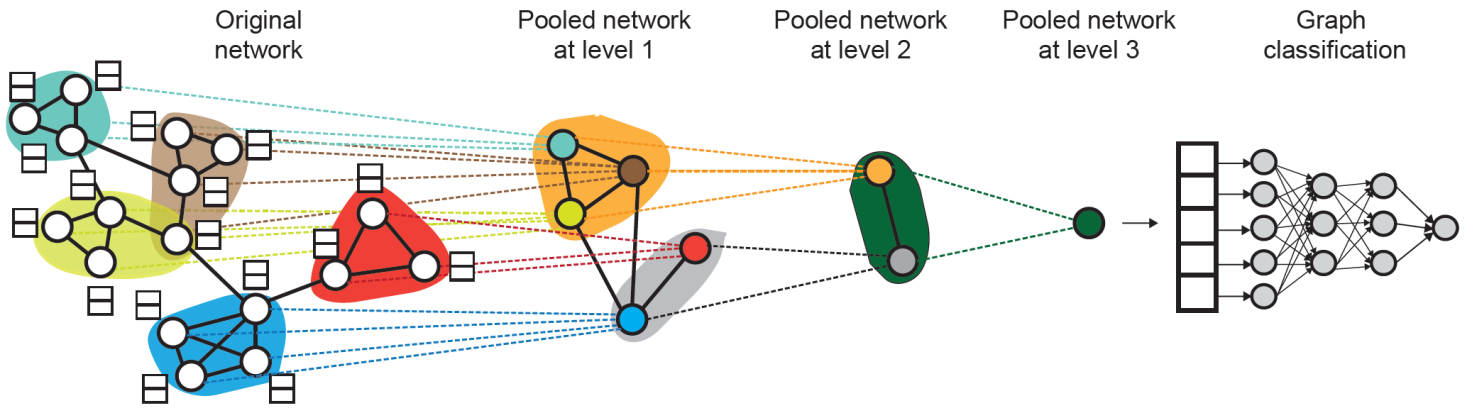
- **DiffPool idea:** 社群分层池化
  - **Hierarchically pool node embeddings**



  - **Leverage 2 independent GNNs at each level**
    - **GNN A:** Compute node embeddings
    - **GNN B:** Compute the cluster that a node belongs to 社群检测(监督学习)
  - **GNNs A and B at each level can be executed in parallel**

Hierarchical Pooling In Practice

- **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**

Stanford CS224W: Training Graph Neural Networks

CS224W: Machine Learning with Graphs

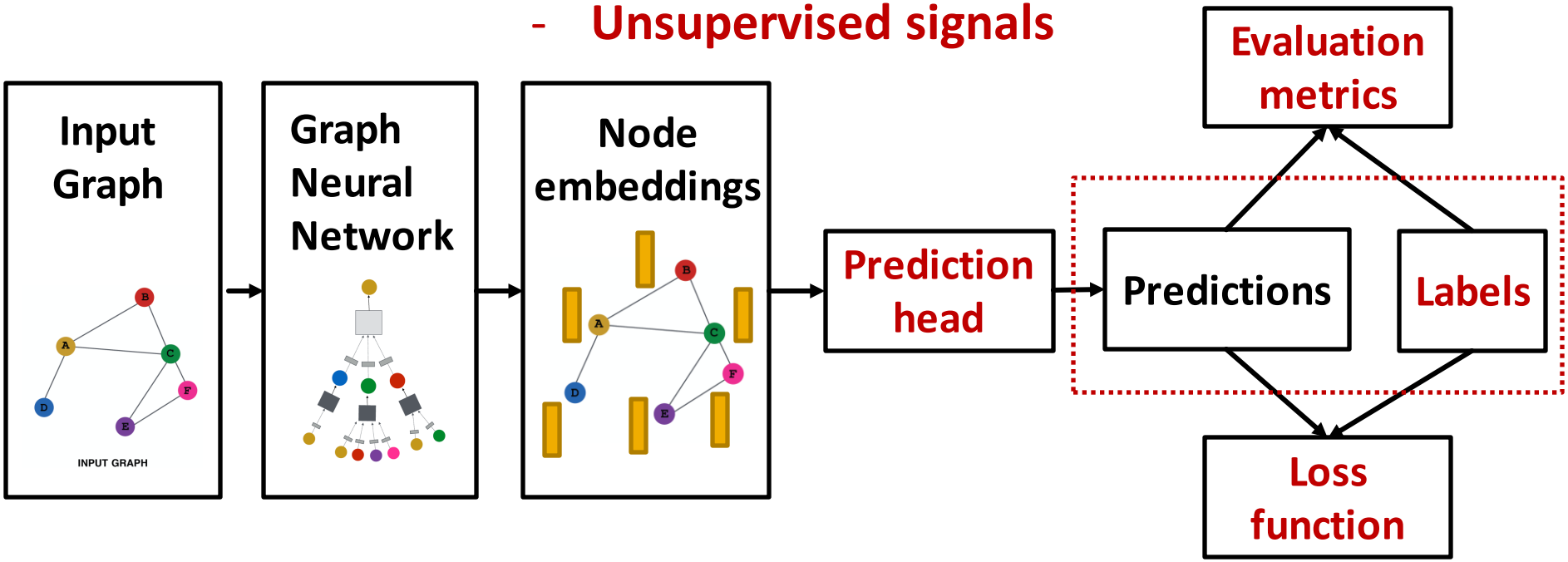
Jure Leskovec, Stanford University

http://cs224w.stanford.edu

GNN Training Pipeline (2)

(2) Where does ground-truth come from?

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



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**"

自监督

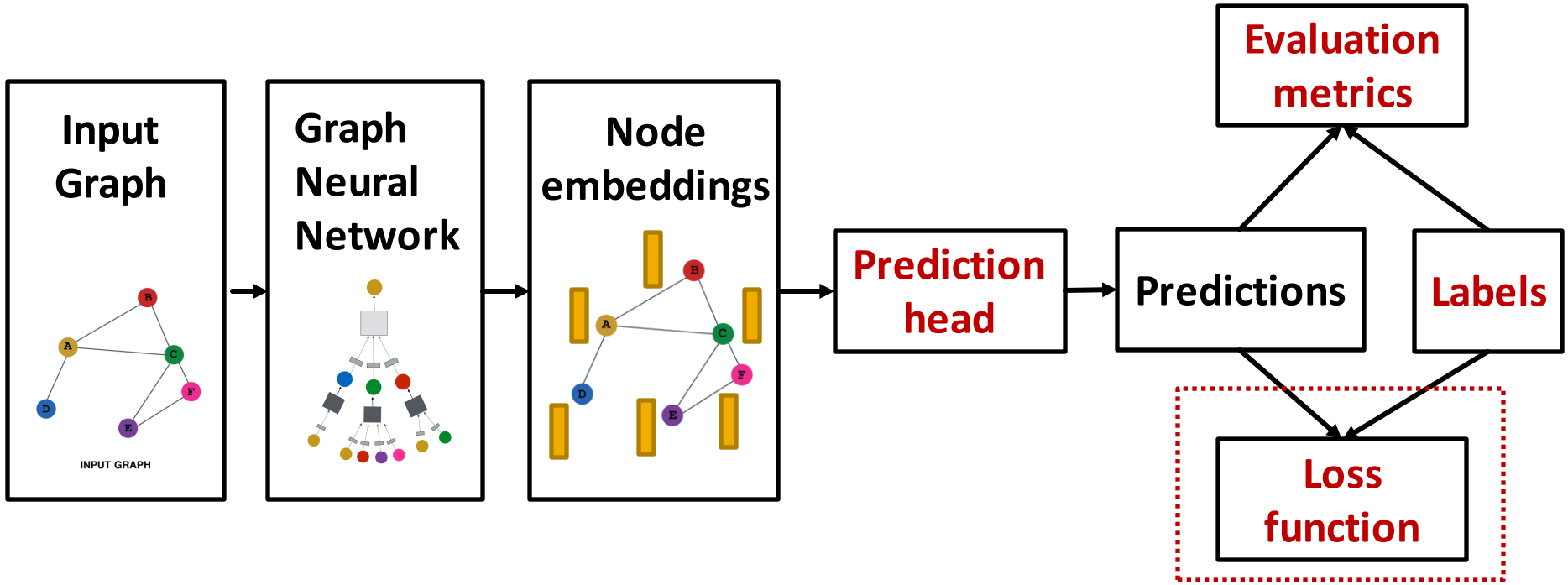
Supervised Labels on Graphs

- **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 学科领域 Cora数据集
  - **Edge labels y_{uv}:** in a transaction network, whether an edge is fraudulent 欺诈
  - **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** DiffPool

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 同形
  - **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 $\hat{\boldsymbol{y}}_v^{(i)}$, label $\boldsymbol{y}_v^{(i)}$
  - **Edge-level**: prediction $\hat{\boldsymbol{y}}_{uv}^{(i)}$, label $\boldsymbol{y}_{uv}^{(i)}$
  - **Graph-level**: prediction $\hat{\boldsymbol{y}}_G^{(i)}$, label $\boldsymbol{y}_G^{(i)}$
  - We will use prediction $\hat{\boldsymbol{y}}^{(i)}$, label $\boldsymbol{y}^{(i)}$ to refer **predictions at all levels**

Classification or Regression

- **Classification**: labels $\boldsymbol{y}^{(i)}$ with discrete value
  - E.g., Node classification: which category does a node belong to
- **Regression**: labels $\boldsymbol{y}^{(i)}$ 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

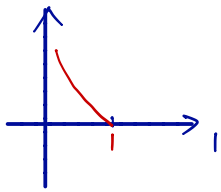
- As discussed in lecture 6, **cross entropy (CE)** is a very common loss function in classification
- K-way prediction for i-th data point:

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

Label: $\boldsymbol{y}^{(i)}$ — Prediction: $\hat{\boldsymbol{y}}^{(i)}$

i-th data point

j-th class



where:

第i个样本

E.g. | 0 | 0 | 1 | 0 | 0 |

$\boldsymbol{y}^{(i)} \in \mathbb{R}^K$ = one-hot label encoding

$\hat{\boldsymbol{y}}^{(i)} \in \mathbb{R}^K$ = prediction after $\mathrm{Softmax}(\cdot)$

E.g. | 0.1 | 0.3 | 0.4 | 0.1 | 0.1 |

$-\log 0.4$

- Total loss over all N training examples

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

Regression Loss

- For regression tasks we often use **Mean Squared Error (MSE)** a.k.a. **L2 loss**

均方误差

- K-way regression for data point (i):

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

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

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

E.g. | 0.9 | 2.8 | 2.0 | 0.3 | 0.8 |

- Total loss over all N training examples

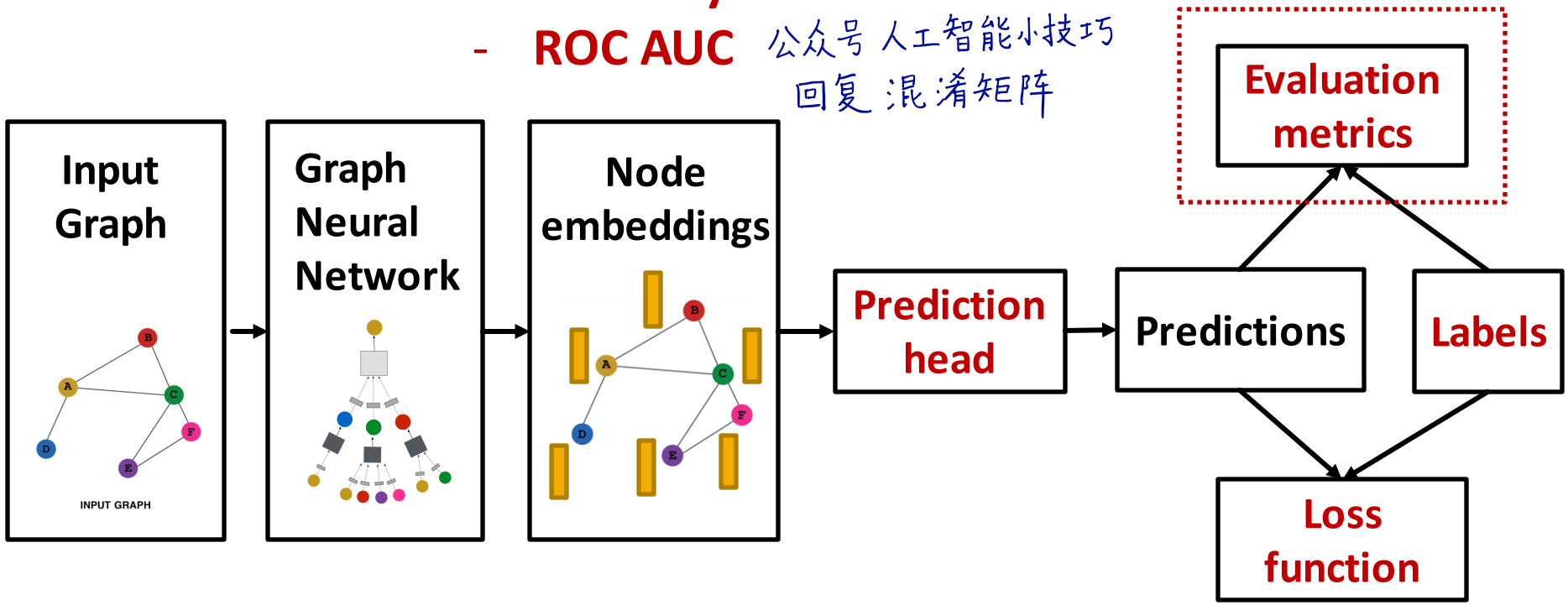
$$\text{Loss} = \sum_{i=1}^{N} \text{MSE}\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right)$$

GNN Training Pipeline (4)

(4) How do we measure the success of a GNN?

- **Accuracy**
- **ROC AUC**

公众号 人工智能小技巧
回复 混淆矩阵



Evaluation Metrics: Regression

- **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)** 均方根误差

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

  - **Mean absolute error (MAE)** 平均绝对误差

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

回归问题的评价指标 见同济子豪兄 波士顿房价数据集视频讲解

Evaluation Metrics: Classification

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

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

公众号 人工智能小技巧
回复 混淆矩阵

- **(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**:

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

公众号 人工智能小技巧
回复 混淆矩阵

- **Precision (P)**:

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

- **Recall (R)**:

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

- **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)

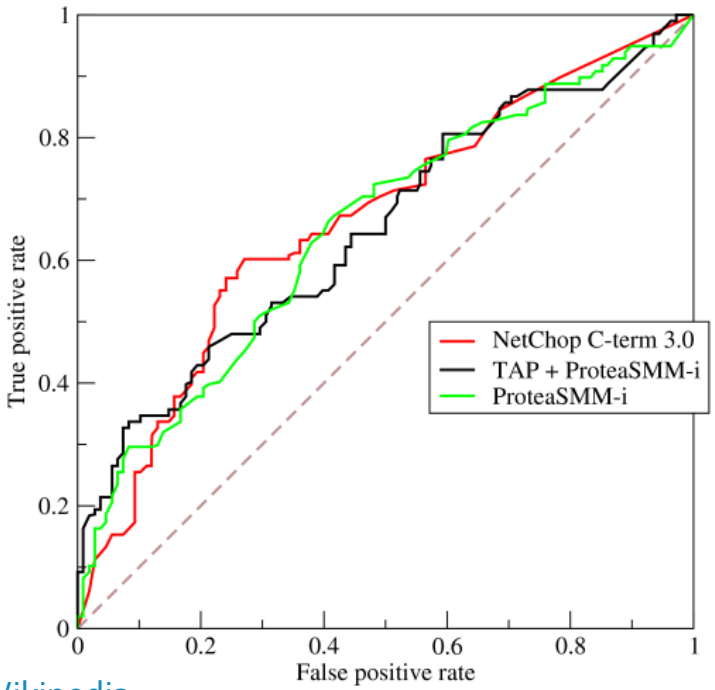
Sklearn Classification Report

(4) Evaluation Metrics

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

公众号 人工智能小技巧
回复 混淆矩阵

TPR



FPR

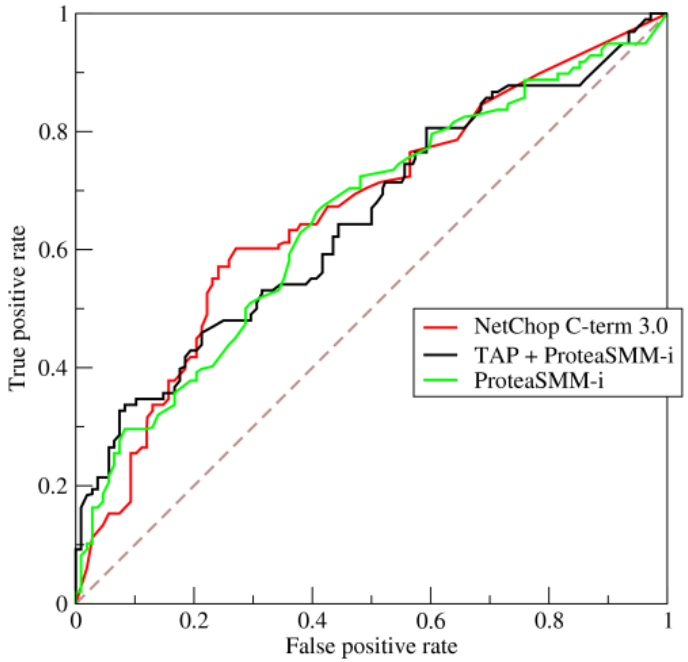
Image Credit: Wikipedia

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

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

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

(4) Evaluation Metrics



公众号 人工智能小技巧
回复 混淆矩阵

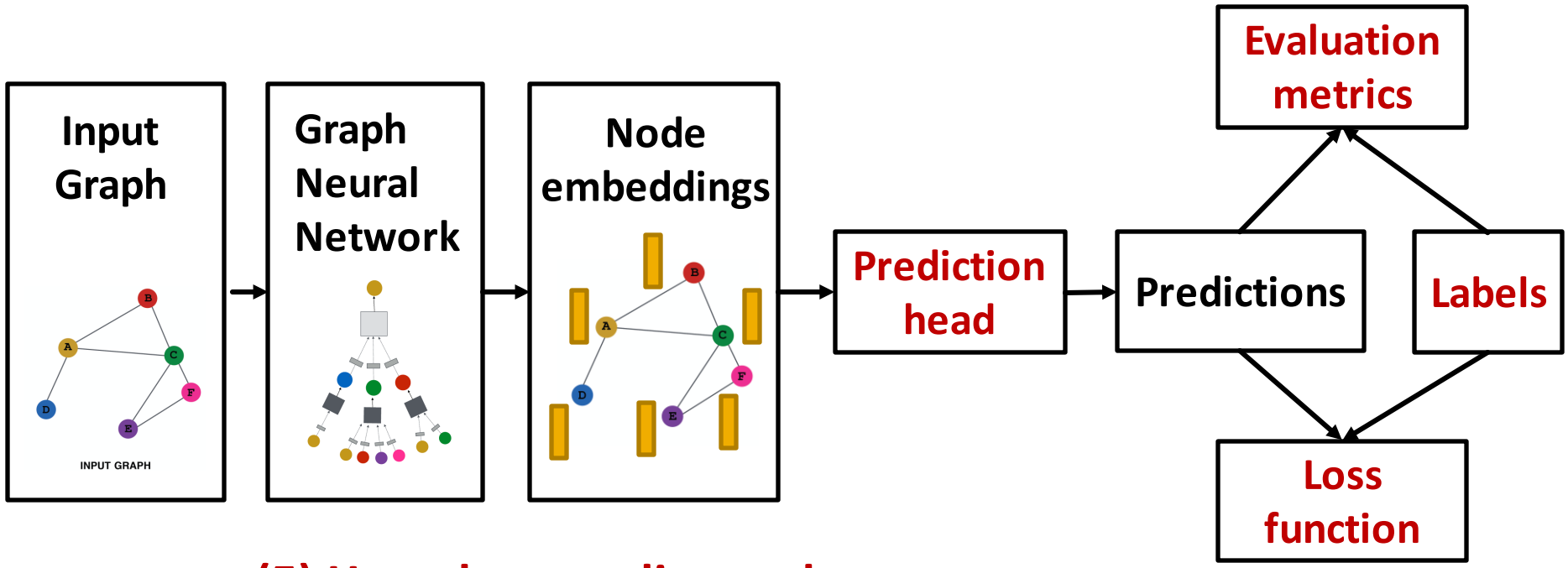
Content Credit: Wikipedia

- **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

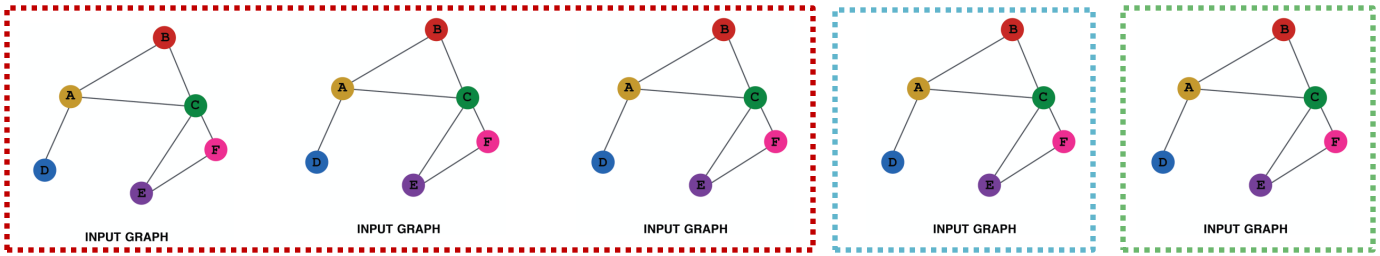
Stanford CS224W: Setting-up GNN Prediction Tasks

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

GNN Training Pipeline (5)



(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
  Data Leakage
  数据泄露
- **Random split:** we will **randomly split** our dataset into training / validation / test
  - We report **average performance over different random seeds**
    类似交叉验证

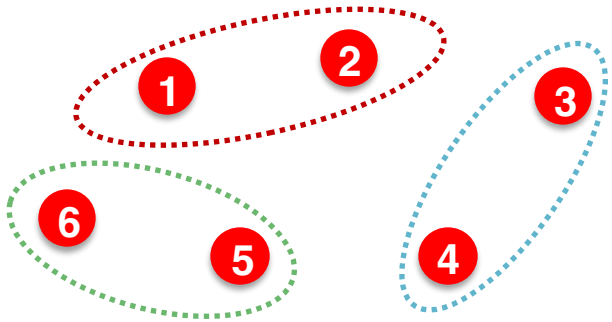
Why Splitting Graphs is Special

- **Suppose we want to split an image dataset**
  - **Image classification:** **Each data point is an image**
  - Here **data points are independent** 样本满足独立同分布 i.i.d. 假设 没有 Data Leakage
    - **Image 5 will not affect our prediction on image 1**

Training

Validation

Test



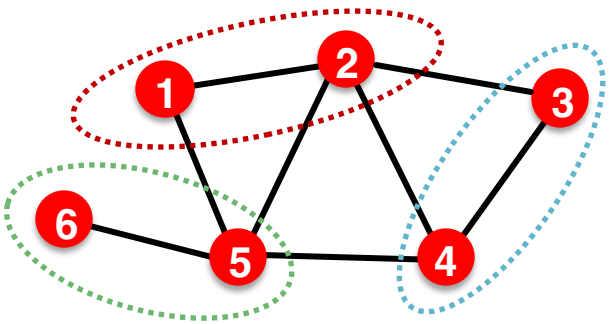
Why Splitting Graphs is Special

- **Splitting a graph dataset is different!**
  - **Node classification: Each data point is a node**
  - Here **data points are NOT independent** 不满足i.i.d.假设
    - **Node 5 will affect our prediction on node 1,** because it will participate in message passing → affect node 1's embedding

Training

Validation

Test



训练集节点的计算图中可能出现测试集节点

- **What are our options?**

Why Splitting Graphs is Special

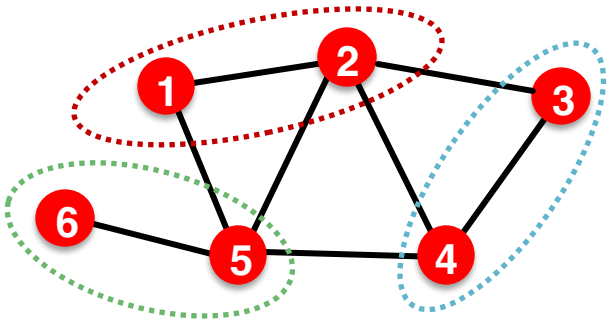
直推式学习

- **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

归纳式学习

- **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**

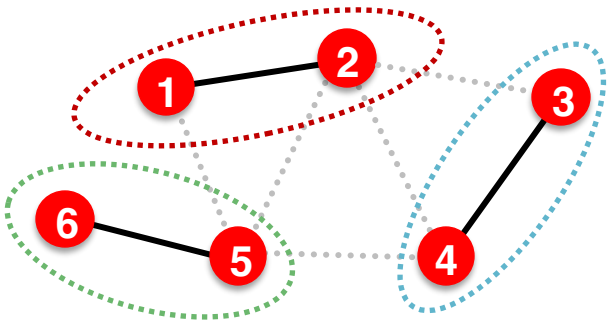
独立无关

  - **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



划分独立的子图数据集
会损失信息

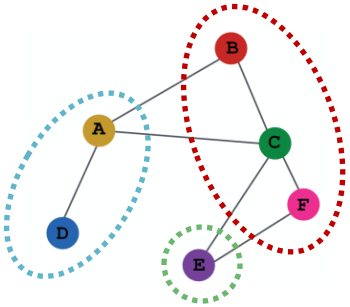
Transductive / Inductive Settings

- **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

泛化到新图

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

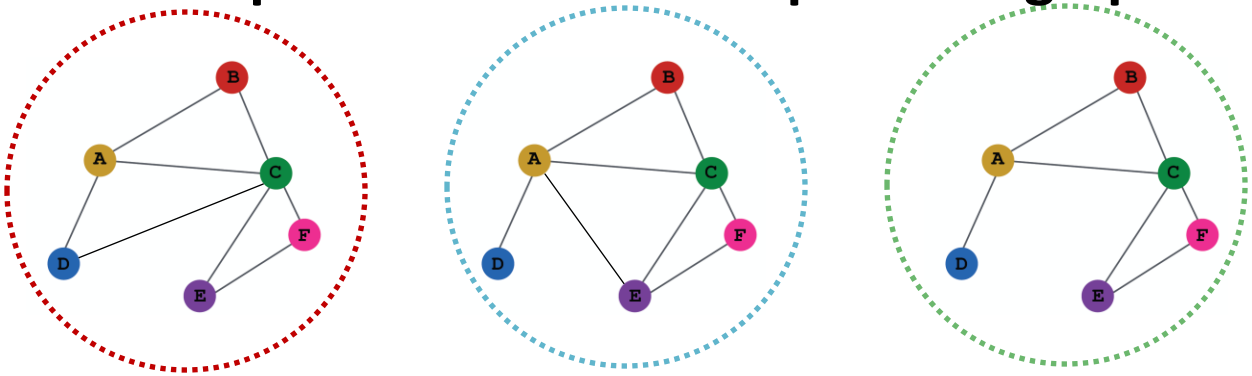


Training

Validation

Test

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



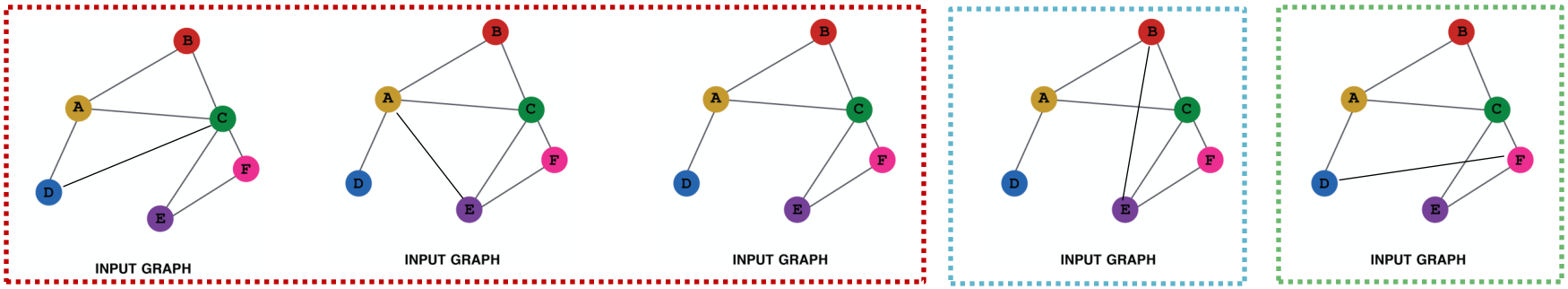
Training

Validation

Test

Example: Graph Classification

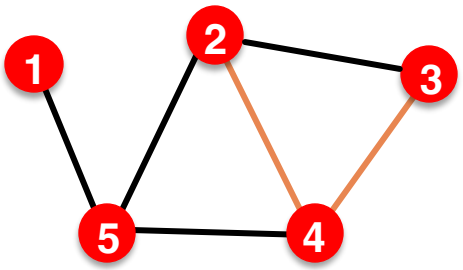
- Only the **inductive setting** is well defined for **graph classification** 全图分类
  - Because **we have to test on unseen graphs**
  - Suppose we have a dataset of 5 graphs. Each split will contain independent graph(s).



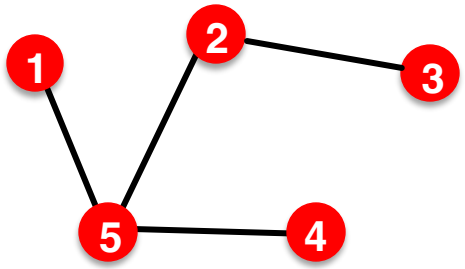
Example: Link Prediction

- **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**

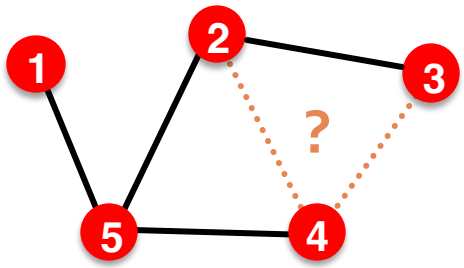
完型填空



Original graph

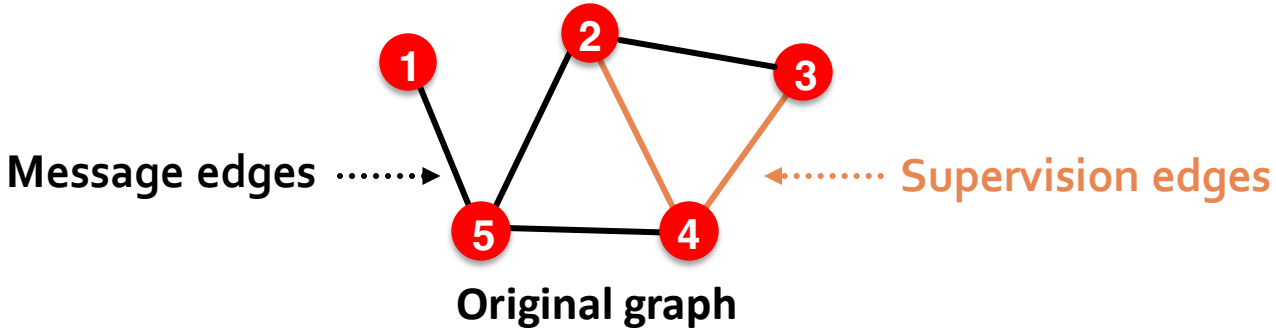


Input graph to GNN



Predictions made by GNN

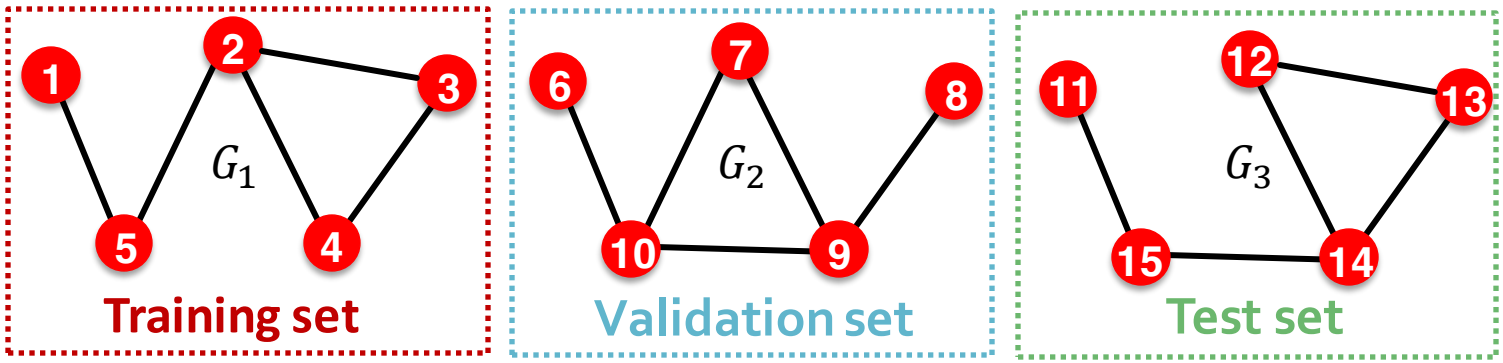
Setting up Link Prediction



- 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!

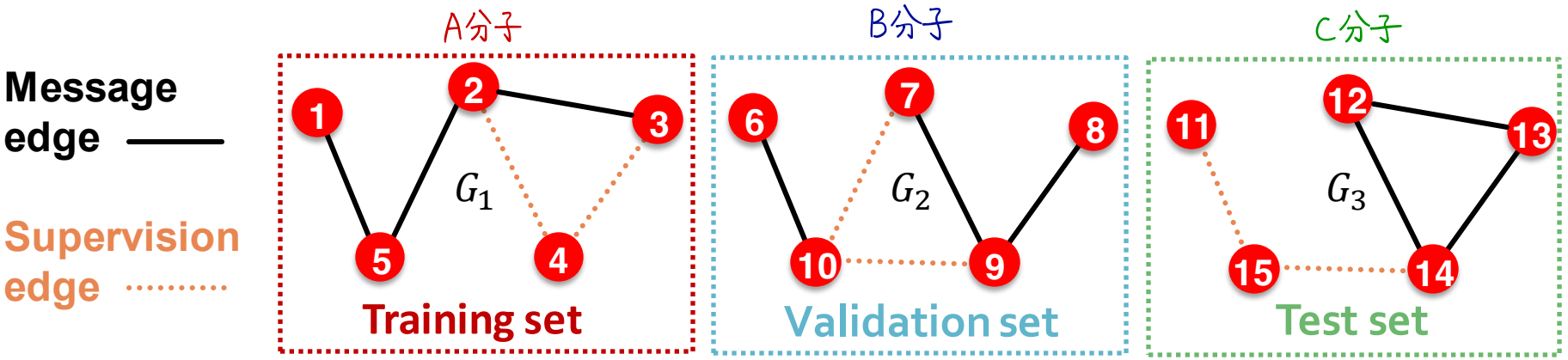
Setting up Link Prediction

- **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**



Setting up Link Prediction

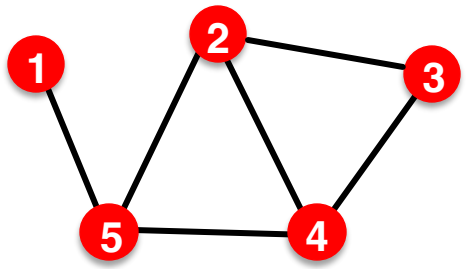
- **Step 2: Split edges into train / validation / test**
- **Option 1: Inductive link prediction split**
  - **Suppose we have a dataset of 3 graphs. Each inductive split will contain an independent graph**
  - **In train or val or test set, each graph will have 2 types of edges: message edges + supervision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



Setting up Link Prediction

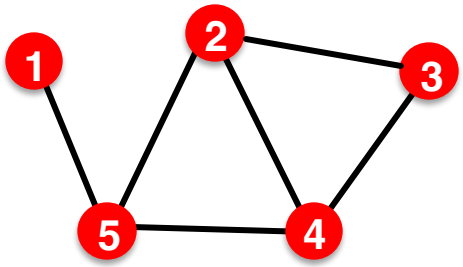
- **Option 2: Transductive link prediction split:**
  - **This is the default setting when people talk about link prediction**
  - **Suppose we have a dataset of 1 graph**

    无法泛化到新图



Setting up Link Prediction

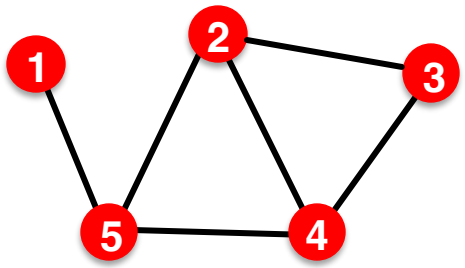
- **Option 2: Transductive link prediction split:**
  - **By definition of "transductive", the entire graph can be observed in all dataset splits**
    - **But since edges are both part of graph structure and the supervision, we need to hold out validation / test edges**
    - **To train the training set, we further need to hold out supervision edges for the training set**



  - **Next:** we will show the exact settings

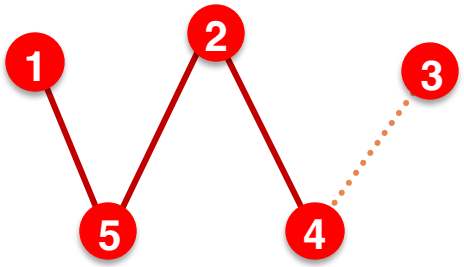
Setting up Link Prediction

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

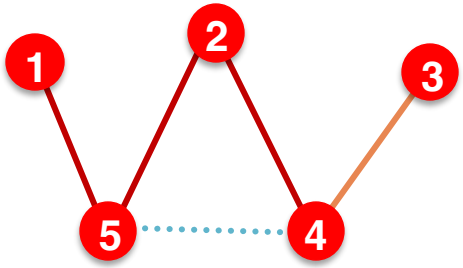


The original graph

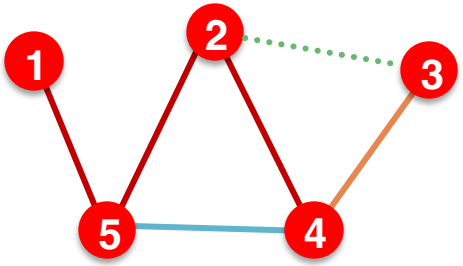
每一阶段都不存在数据泄露



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



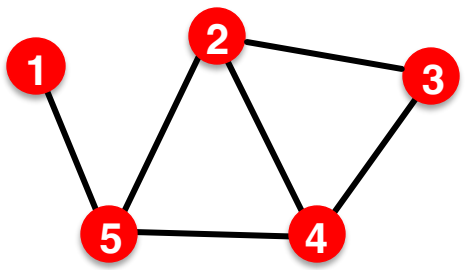
(2) At validation time:
Use **training message edges & training supervision edges** to predict **validation edges**



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

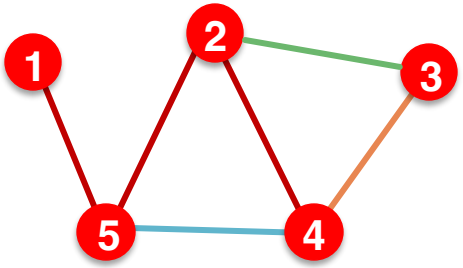
Setting up Link Prediction

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



The original graph

Split

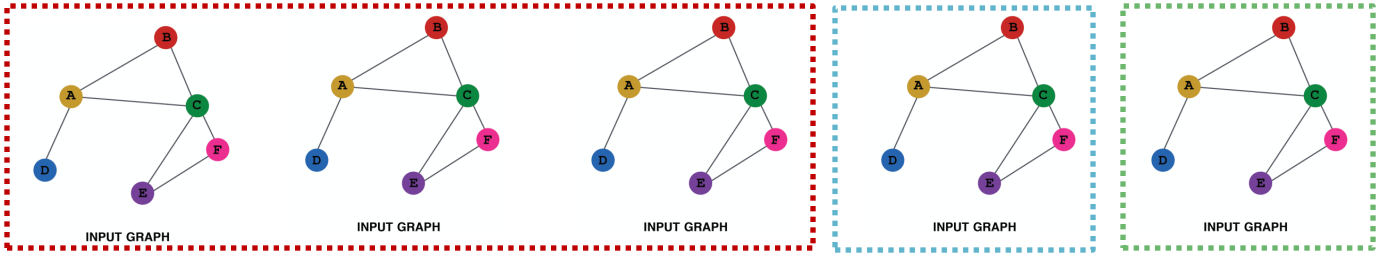


Split Graph with
4 types of edges

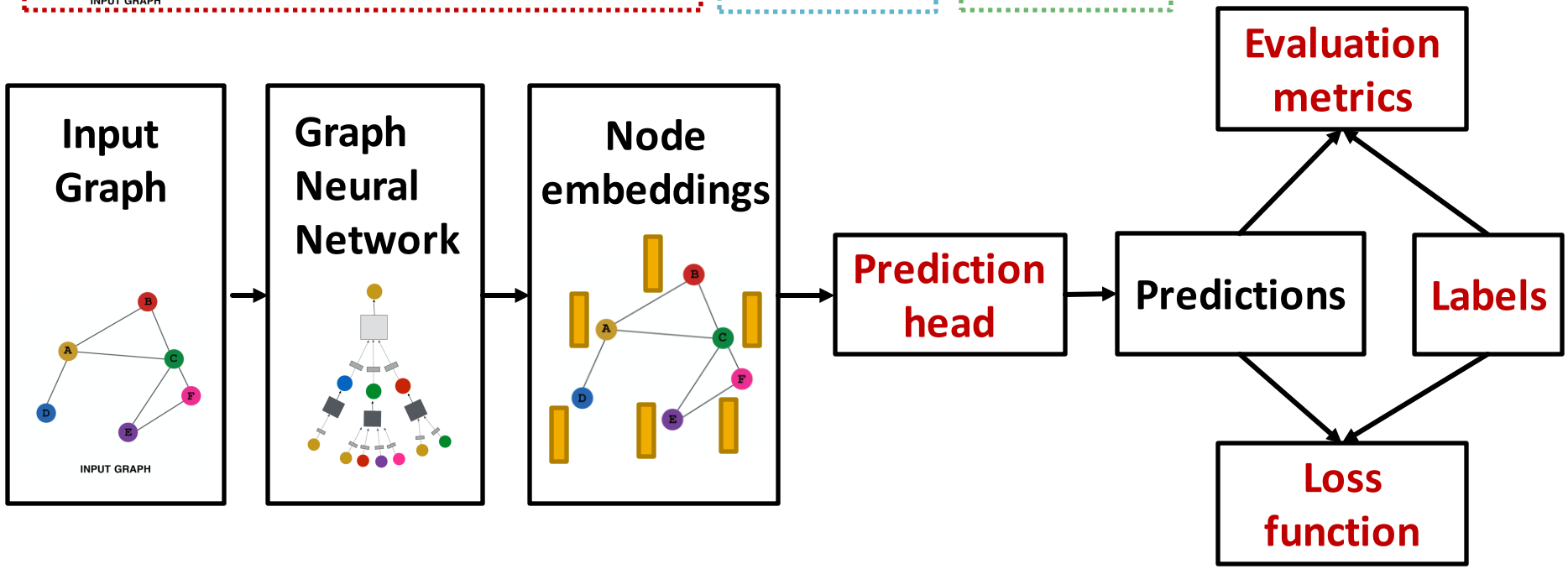
划分四种 Edge
Training message edges
Training supervision edges
Validation edges
Test edges

- **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



Implementation resources:

DeepSNAP provides core modules for this pipeline

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