Training Graph Neural Network

ACMS 80770: Deep Learning with Graphs

Instructor: Navid Shervani-Tabar

Department of Applied and Comp Math and Stats

Training GNNs

- In the previous lectures, we discussed the **propagation rule** for the neural message passing algorithm and how to build GNN models.
- We further reviewed different approaches to **enhance** the performance of these models.
- In this lecture, we will focus on **training** these architectures to carry out **tasks** on different levels on graphs.

Training GNNs

- In the previous lectures, we discussed the **propagation rule** for the neural message passing algorithm and how to build GNN models.
- We further reviewed different approaches to **enhance** the performance of these models.
- In this lecture, we will focus on **training** these architectures to carry out **tasks** on different levels on graphs.
- Fulfilling this objective requires addressing the following **questions**:

Training GNNs

- In the previous lectures, we discussed the **propagation rule** for the neural message passing algorithm and how to build GNN models.
- We further reviewed different approaches to **enhance** the performance of these models.
- In this lecture, we will focus on **training** these architectures to carry out **tasks** on different levels on graphs.
- Fulfilling this objective requires addressing the following **questions**:
  - How to turn **embeddings to predictions** for the problem of interest?

Training GNNs

- In the previous lectures, we discussed the **propagation rule** for the neural message passing algorithm and how to build GNN models.
- We further reviewed different approaches to **enhance** the performance of these models.
- In this lecture, we will focus on **training** these architectures to carry out **tasks** on different levels on graphs.
- Fulfilling this objective requires addressing the following **questions**:
  - How to turn **embeddings to predictions** for the problem of interest?
  - How to **define a loss** function over the predicted values to evaluate and train the model?

Node classification

- One of the most popular graph-based problems is **node classification**.
- In this problem, we are **given node embeddings** $\boldsymbol{z}_i \in \mathbb{R}^d$ and we want to **predict the category** c to which node v_i belongs.

Node classification

- One of the most popular graph-based problems is **node classification**.
- In this problem, we are **given node embeddings** $\boldsymbol{z}_i \in \mathbb{R}^d$ and we want to **predict the category** c to which node v_i belongs.
  - Categorizing papers into research topics based on the citation graph and words used in the paper.

Node classification

- One of the most popular graph-based problems is **node classification**.
- In this problem, we are **given node embeddings** $\boldsymbol{z_i} \in \mathbb{R}^d$ and we want to **predict the category** c to which node v_i belongs.
  - Categorizing papers into research topics based on the citation graph and words used in the paper.
- We can use a linear transformation to **map $\boldsymbol{z_i}$ to a score vector** that represents the likelihood of v_i belonging to each category.
- Mathematically put,

$$\bar{\boldsymbol{y}} = f(\boldsymbol{z}_i) = \boldsymbol{z}_i \mathbf{W}$$

where $\mathbf{W} \in \mathbb{R}^{d \times C}$ is learnable parameter.

Node classification

- We can turn $\bar{\boldsymbol{y}}$ into a **vector of probabilities** using the softmax function

$$[\hat{\boldsymbol{y}}_i]_c = \text{softmax}\left([\bar{\boldsymbol{y}}_i]_c\right) = \frac{\exp\left([\bar{\boldsymbol{y}}_i]_c\right)}{\sum_{j=1}^{C} \exp\left([\bar{\boldsymbol{y}}_i]_j\right)}$$

Node classification

- We can turn $\bar{\boldsymbol{y}}$ into a **vector of probabilities** using the softmax function

$$[\hat{\boldsymbol{y}}_i]_c = \text{softmax}\left([\bar{\boldsymbol{y}}_i]_c\right) = \frac{\exp\left([\bar{\boldsymbol{y}}_i]_c\right)}{\sum_{j=1}^{C} \exp\left([\bar{\boldsymbol{y}}_i]_j\right)}$$

- We can then define the **loss function** as

$$\mathcal{L} = \sum_{v_i \in V_{train}} -\log\left(p\left(\boldsymbol{y}_i \mid \boldsymbol{z}_i\right)\right)$$

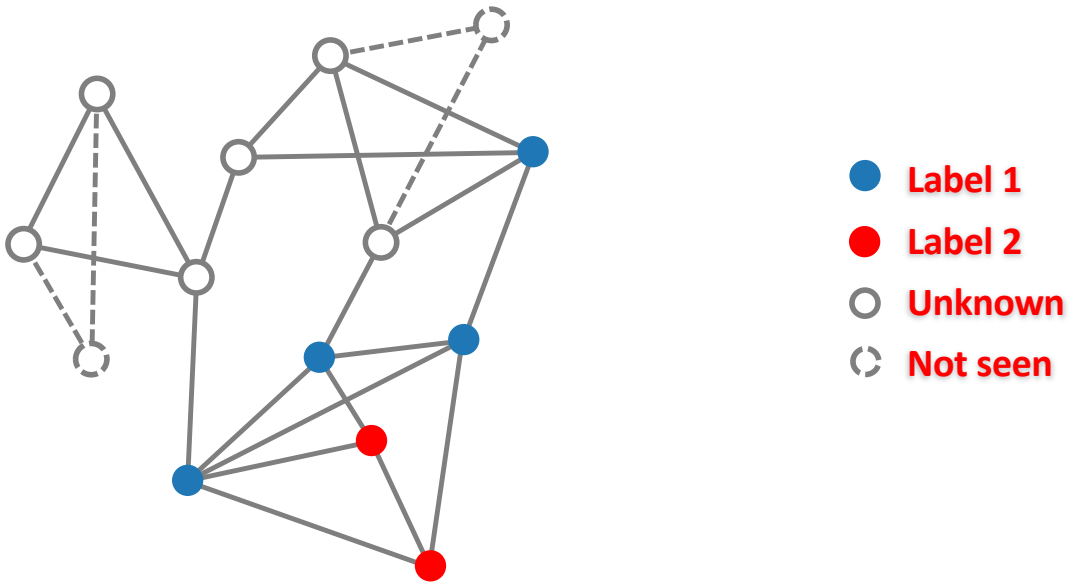
where V_{train} represents the set of training nodes and

$$p\left(\boldsymbol{y}_i \mid \boldsymbol{z}_i\right) = \sum_{c=1}^{C} [\boldsymbol{y}_i]_c \, [\hat{\boldsymbol{y}}_i]_c$$

with $\boldsymbol{y}_i$ representing the one-hot vector of **true label**.

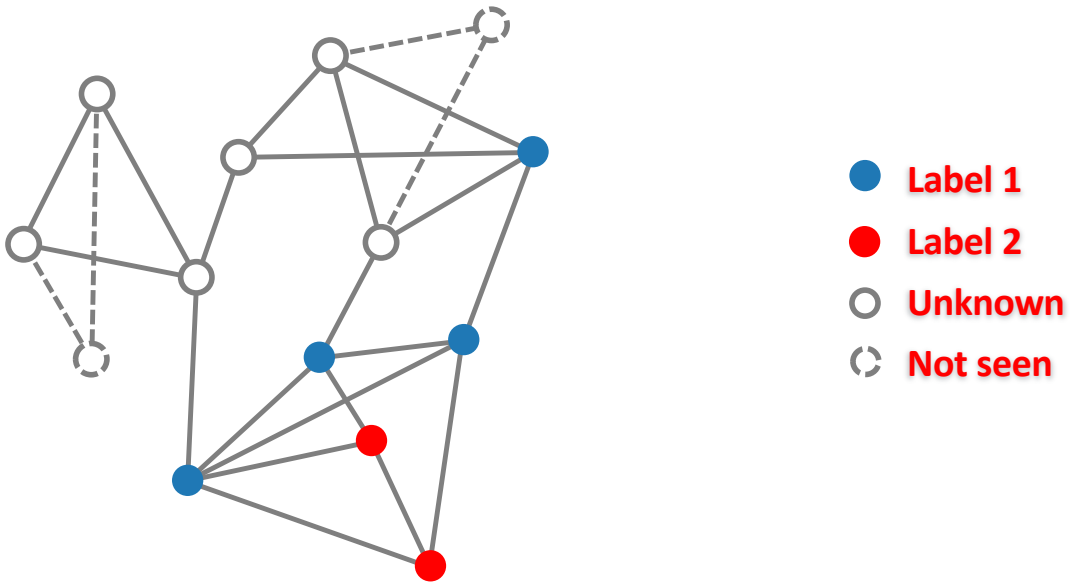
Training and Test Nodes

- The **data provided** for this task can be divided into **three** categories:



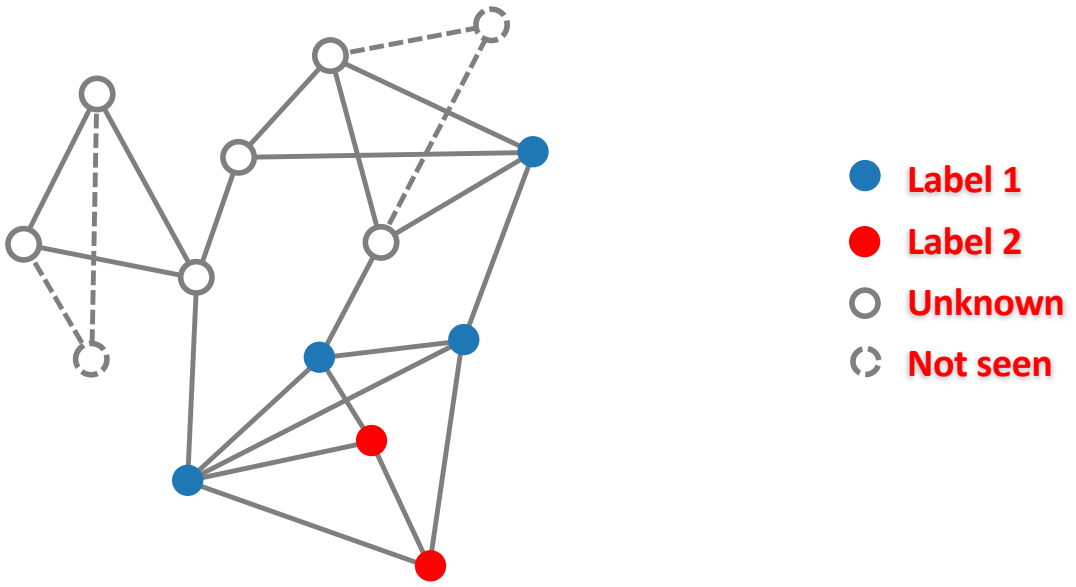
Training and Test Nodes

- The **data provided** for this task can be divided into **three** categories:
  - Training nodes V_{train} (Label 1 and Label 2).



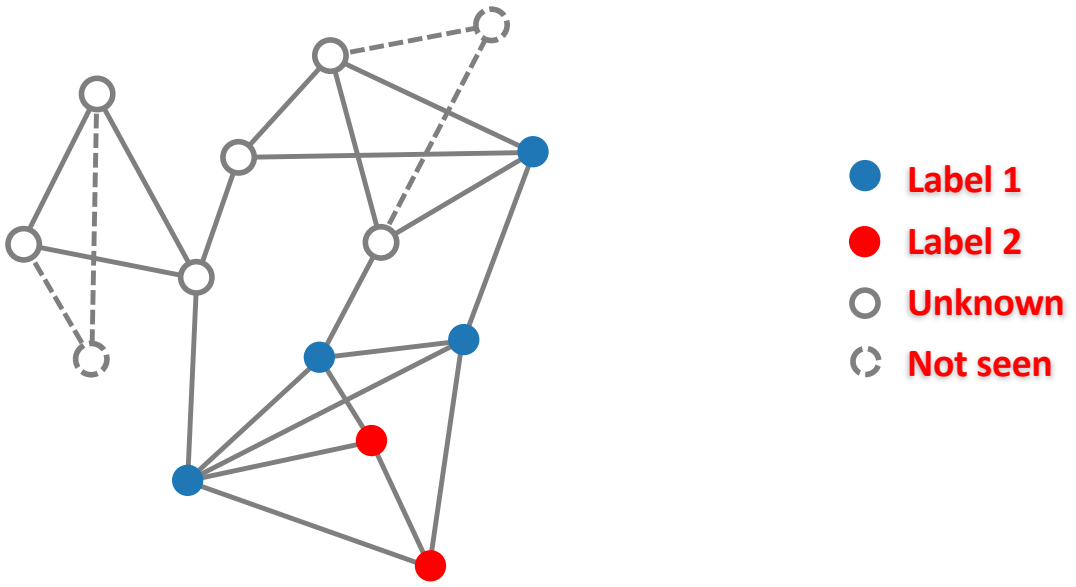
Training and Test Nodes

- The **data provided** for this task can be divided into **three** categories:
  - Training nodes V_{train} (Label 1 and Label 2).
  - Transductive test nodes V_{test}^{tr} (Unknown Label).



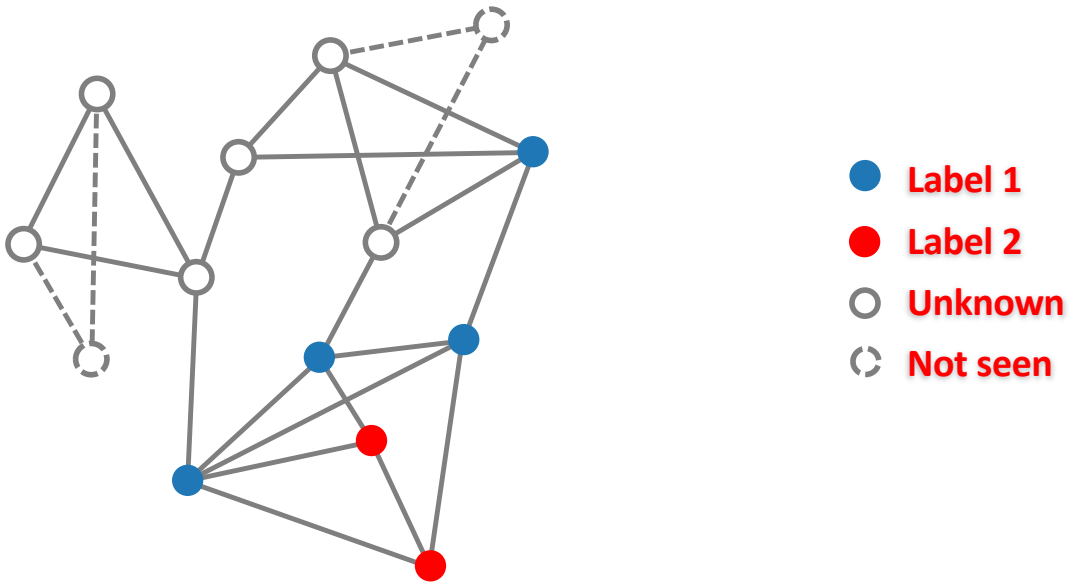
Training and Test Nodes

- The **data provided** for this task can be divided into **three** categories:
  - Training nodes V_{train} (Label 1 and Label 2).
  - Transductive test nodes V_{test}^{tr} (Unknown Label).
  - Inductive test nodes V_{test}^{in} (Not seen).



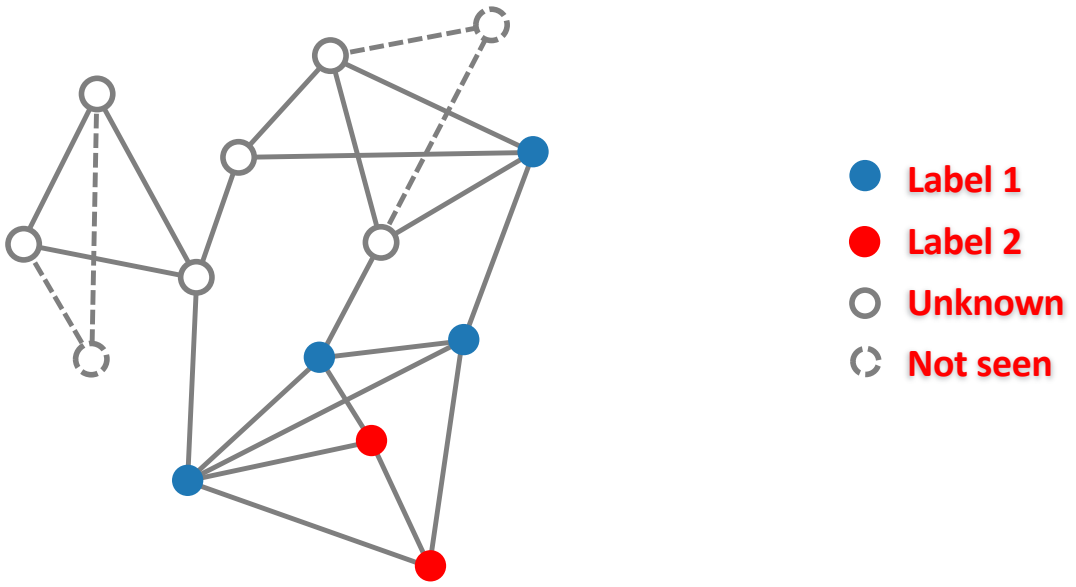
Training and Test Nodes

- **Training nodes** V_{train}:
  - The **loss** function $\mathcal{L}$ **is computed** over the set of training nodes V_{train}.
  - During the training, the nodes $v_i \in V_{train}$ **participate** in the **message passing** step.



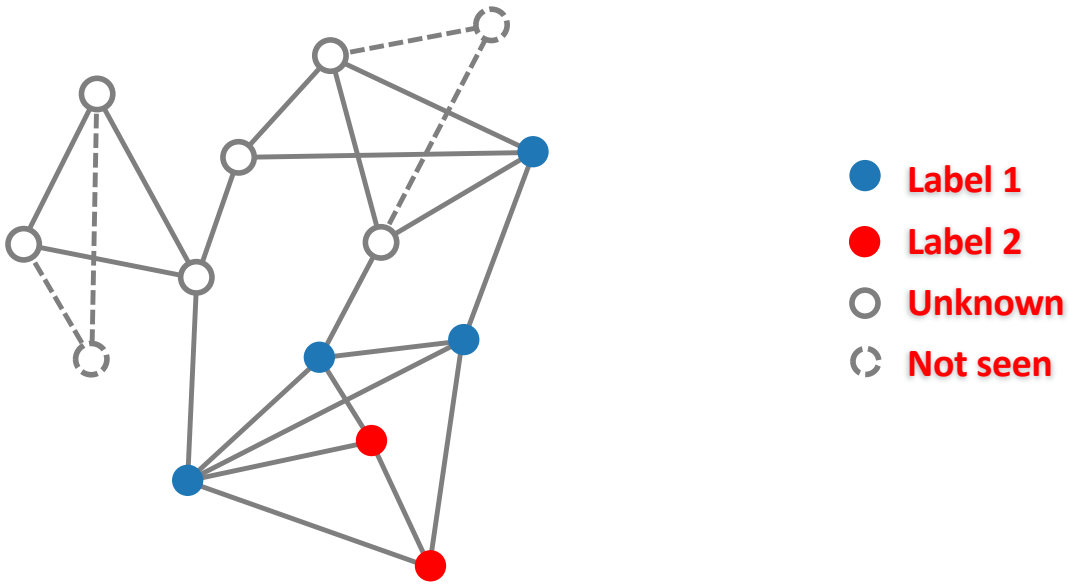
Training and Test Nodes

- **Transductive test nodes** V_{test}^{tr}:
  - The nodes $v_i \in V_{test}^{tr}$ are present during the training and **participate** in the **message passing** process.
  - However, labels for $v_i \in V_{test}^{tr}$ are unknown, and hence, their embedding is **not** participating in the **loss** calculation.



Training and Test Nodes

- **Inductive test nodes** V_{test}^{in}:
  - The nodes $v_i \in V_{test}^{in}$ are **not seen** during training and are masked along with their incident edge.
  - As a result, the nodes $v_i \in V_{test}^{in}$ participate **neither** in **message passing**, **nor** in computation of the **loss**.

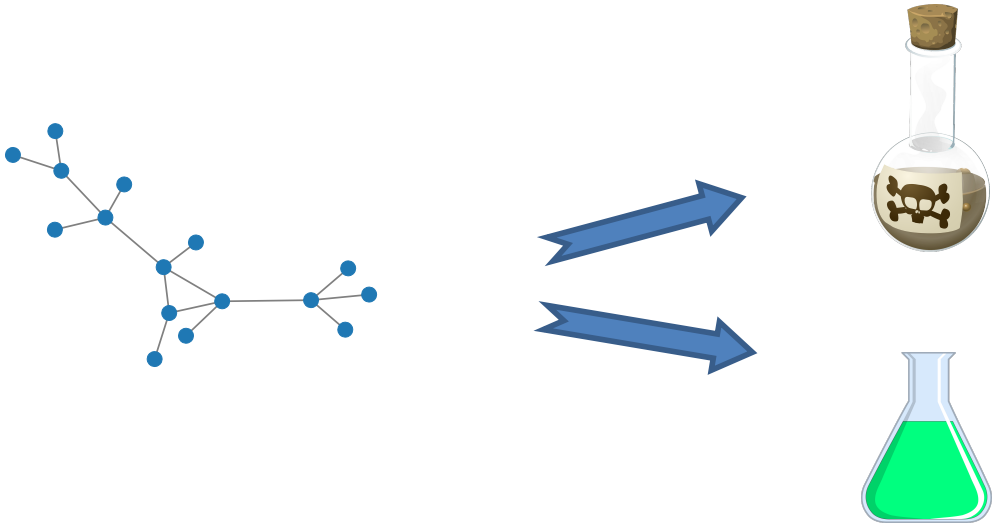


Graph-level Classification

- Another group of popular graph problems involve **graph-level classification**.

Graph-level Classification

- Another group of popular graph problems involve **graph-level classification**.
  - Solubility of molecular graphs.



Graph-level Classification

- Another group of popular graph problems involve **graph-level classification**.
  - Solubility of molecular graphs.
- In this problem, we assign the graph to one of the different possible **categories** based on its graph-level **embedding** $\boldsymbol{z}_G$.
- Similar to node classification, we **map** graph-level representation $\boldsymbol{z}_G$ to a **set of scores** over different possible categories

$$\bar{\boldsymbol{y}} = \boldsymbol{z}_G \mathbf{W}$$

where $\mathbf{W} \in \mathbb{R}^{d \times C}$ is a trainable parameter.

Graph-level Classification

- Then, we use a softmax function to turn these scores into **probability vectors**.

$$[\hat{\boldsymbol{y}}_i]_c = \text{softmax}\left([\bar{\boldsymbol{y}}_i]_c\right) = \frac{\exp\left([\bar{\boldsymbol{y}}_i]_c\right)}{\sum_{j=1}^{C} \exp\left([\bar{\boldsymbol{y}}_i]_j\right)}$$

- Then, we compute the **loss** as

$$\mathcal{L} = \sum_{G_i \in \mathcal{G}} -\log\left(p\left(\boldsymbol{y}_i \mid \boldsymbol{z}_{G_i}\right)\right)$$

where $\mathcal{G} = \{G_1, \ldots, G_N\}$ is the set of N training graphs and

$$p\left(\boldsymbol{y}_i \mid \boldsymbol{z}_{G_i}\right) = \sum_{c=1}^{C} [\boldsymbol{y}_i]_c \, [\hat{\boldsymbol{y}}_i]_c$$

Graph-level Regression

- Another popular problem is graph-level **regression problem**.
- Given the embedding of the graph, we predict the target value using

$$\hat{\boldsymbol{y}} = \boldsymbol{z}_{G_i}\mathbf{W}$$

where $\mathbf{W} \in \mathbb{R}^d$ is a vector of trainable parameters.

Graph-level Regression

- Another popular problem is graph-level **regression problem**.
- Given the embedding of the graph, we predict the target value using

$$\hat{\boldsymbol{y}} = \boldsymbol{z}_{G_i}\mathbf{W}$$

  where $\mathbf{W} \in \mathbb{R}^d$ is a vector of trainable parameters.
- Then, we compute the **loss function** as

$$\mathcal{L} = \sum_{G_i \in \mathcal{G}} \|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i\|_2^2$$

- In principle, both classification and regression in the graph-level can use **deeper MLPs** as instead of a linear transformation to map $\boldsymbol{z}_G$ into target labels.

Computational Efficiency

- Performing neural message passing using **node-level** implementations can be computationally **inefficient**.
- One approach that alleviates this is to use **graph-level equations** to perform neural message passing.

Computational Efficiency

- Performing neural message passing using **node-level** implementations can be computationally **inefficient**.
- One approach that alleviates this is to use **graph-level equations** to perform neural message passing.
- GIN

$$\boldsymbol{h}_i^{(k)} = \text{MLP}^{(k-1)}\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{h}_i^{(k-1)} + \sum_{v_j \in N(v_i)} \boldsymbol{h}_j^{(k-1)}\right)$$

$$\boldsymbol{H}^{(k)} = \text{MLP}^{(k-1)}\left(\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{I} + \mathbf{A}\right)\boldsymbol{H}^{(k-1)}\right)$$

Computational Efficiency

- Performing neural message passing using **node-level** implementations can be computationally **inefficient**.
- One approach that alleviates this is to use **graph-level equations** to perform neural message passing.
- GIN

$$\boldsymbol{h}_i^{(k)} = \text{MLP}^{(k-1)}\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{h}_i^{(k-1)} + \sum_{v_j \in N(v_i)} \boldsymbol{h}_j^{(k-1)}\right)$$

$$\boldsymbol{H}^{(k)} = \text{MLP}^{(k-1)}\left(\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{I} + \mathbf{A}\right)\boldsymbol{H}^{(k-1)}\right)$$

- Graph-level implementation, however, has some **drawbacks.**

Computational Efficiency

- Performing neural message passing using **node-level** implementations can be computationally **inefficient**.
- One approach that alleviates this is to use **graph-level equations** to perform neural message passing.
  - GIN

$$\boldsymbol{h}_i^{(k)} = \text{MLP}^{(k-1)}\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{h}_i^{(k-1)} + \sum_{v_j \in N(v_i)} \boldsymbol{h}_j^{(k-1)}\right)$$

$$\boldsymbol{H}^{(k)} = \text{MLP}^{(k-1)}\left(\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{I} + \mathbf{A}\right)\boldsymbol{H}^{(k-1)}\right)$$

- Graph-level implementation, however, has some **drawbacks.**
  - For larger graphs, we might not have **sufficient memory** available to operate on the graph-level.

Computational Efficiency

- Performing neural message passing using **node-level** implementations can be computationally **inefficient**.
- One approach that alleviates this is to use **graph-level equations** to perform neural message passing.
- GIN

$$\boldsymbol{h}_i^{(k)} = \text{MLP}^{(k-1)}\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{h}_i^{(k-1)} + \sum_{v_j \in N(v_i)} \boldsymbol{h}_j^{(k-1)}\right)$$

$$\boldsymbol{H}^{(k)} = \text{MLP}^{(k-1)}\left(\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{I} + \mathbf{A}\right)\boldsymbol{H}^{(k-1)}\right)$$

- Graph-level implementation, however, has some **drawbacks.**
  - For larger graphs, we might not have **sufficient memory** available to operate on the graph-level.
  - Using graph-level computations **prevents** us from using **mini-batches** while training the model.

Computational Efficiency

- Another approach is using **mini-batches** to train the model.
- We divide each training iteration (epoch) into **a few mini-batches** containing subset of the nodes in the graph.
- At each epoch, we iterate over the mini-batches and only perform **message passing** over the **nodes in the mini-batch**.

Computational Efficiency

- Another approach is using **mini-batches** to train the model.
- We divide each training iteration (epoch) into **a few mini-batches** containing subset of the nodes in the graph.
- At each epoch, we iterate over the mini-batches and only perform **message passing** over the **nodes in the mini-batch**.
- However, there are some caveats with this approach

Computational Efficiency

- Another approach is using **mini-batches** to train the model.
- We divide each training iteration (epoch) into **a few mini-batches** containing subset of the nodes in the graph.
- At each epoch, we iterate over the mini-batches and only perform **message passing** over the **nodes in the mini-batch**.
- However, there are some caveats with this approach
  - Removing the rest of nodes at each mini-batch results in **loss of information**.

Computational Efficiency

- Another approach is using **mini-batches** to train the model.
- We divide each training iteration (epoch) into **a few mini-batches** containing subset of the nodes in the graph.
- At each epoch, we iterate over the mini-batches and only perform **message passing** over the **nodes in the mini-batch**.
- However, there are some caveats with this approach
  - Removing the rest of nodes at each mini-batch results in **loss of information**.
  - Removing nodes may also result in **disconnecting** the graph.

Computational Efficiency

- Another approach is using **mini-batches** to train the model.
- We divide each training iteration (epoch) into **a few mini-batches** containing subset of the nodes in the graph.
- At each epoch, we iterate over the mini-batches and only perform **message passing** over the **nodes in the mini-batch**.
- However, there are some caveats with this approach
  - Removing the rest of nodes at each mini-batch results in **loss of information**.
  - Removing nodes may also result in **disconnecting** the graph.
- Therefore, **extra care** needs to be given when training large graphs in mini-batches.

Summary

- Node classification
- Dataset
  - Training nodes
  - Transductive test nodes
  - Inductive test nodes
- Graph-level Classification
- Graph-level Regression
- Computational Efficiency