We ask you to develop a tutorial/case-study on applying state-of-the-art graph ML to a real-world problem using PyG.

- Someone can then follow your tutorial and learn how to apply ML to a real-world problem.

The project is open-ended: You can choose models and problems to work on.

- In the instruction doc, we provide some examples of models and datasets.

The final product will be a draft blog post that you share with us privately.

- With your permission we would publish best ones at PyG.org.

Groups of 3 students are strongly recommended; groups of 1-2 also permitted.
Why Blog Posts?

- A great exercise for you to understand and implement *graph ML models applied to real-world problems*.
- A great lasting resource for the broader community to study graph ML.
  - Blog posts are more accessible than technical reports.
  - We will publicize selected blog posts from you!
What are Good Blog Posts?

- **Good blog posts should include**
  - **Step-by-step explanation of graph ML techniques**
    - Assume your readers are
      - familiar with ML, deep learning, and Pytorch
      - not familiar with graph ML and PyG
  - **Visualization**
    - To explain techniques and results, Gifs > Images > Text
    - The more visualization, the better.
  - **Code snippets of PyG/Pytorch**
  - **Link to Google Colab to reproduce your results**
    - Your Colab should be readable and include enough documentations.
Application Domains

Application domains of graph ML includes:

- Recommender systems
- Molecule classification
- Paper classification in citation networks
- Knowledge graph completion
- Product classification in co-purchasing graphs
- Fraud detection in transaction networks
- Protein function prediction in protein-protein interaction networks
- Friend recommendation in social networks
Finding Graph ML Models

- **OGB Leaderboard**
  - [https://ogb.stanford.edu/docs/leader_overview/](https://ogb.stanford.edu/docs/leader_overview/)

- **Top ML conference papers:**
  - KDD
  - ICLR
  - ICML
  - NeurIPS
  - WWW
  - WSDM

**Tips:** Narrow down relevant papers by searching titles (e.g., containing “graph”).
By October 19, 11:59pm PT
The proposal should include the following:

- **Application domain**
  - Which dataset are you planning to use?
  - Describe the dataset, prediction tasks, and metric.
  - Why did you choose the dataset?

- **Graph ML technique that you want to apply**
  - Graph ML model you plan to use
  - Describe the model (using figures and equations)
  - Why the model is appropriate for the dataset?
Special OH this week dedicated to project

- Jure’s OH: 1-3pm on Wed 10/13
  - 10 min slots: https://calendly.com/cs224w-oh/jure-project
- Weihua’s OH: 10am-12pm on Thu 10/14
  - 15 min slots: https://calendly.com/cs224w-oh/weihua-project
  - This will be recurring every Thursday

How to sign up

- One person from the group should sign up and add their group members under "Guest emails"
- Zoom link will be in the invite – you will be let off the waiting room when it is your turn, be on time!
Stanford CS224W: A General Perspective on Graph Neural Networks

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

Output: Node embeddings. Also, we can embed subgraphs, graphs.
Idea: Node’s neighborhood defines a computation graph

Learn how to propagate information across the graph to compute node features
**Intuition:** Nodes aggregate information from their neighbors using neural networks.
**Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!
Why GNNs generalize other NNs?

- Defined notions of permutation invariance and equivariance.

- How does GNNs compare to prominent architectures such as Convolutional Neural Nets, and Transformers?
Convolutional Neural Network

Convolutional neural network (CNN) layer with 3x3 filter:

\[
h_v^{(l+1)} = \sigma(\sum_{u \in N(v) \cup \{v\}} W_u^l h_u^{(l)}), \quad \forall l \in \{0, ..., L - 1\}
\]

\(N(v)\) represents the 8 neighbor pixels of \(v\).
GNN vs. CNN

Convolutional neural network (CNN) layer with 3x3 filter:

- GNN formulation (previous slide): $h_v^{(l+1)} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}), \forall l \in \{0, ..., L - 1\}$
- CNN formulation:
  - if we rewrite:
    $h_v^{(l+1)} = \sigma(\sum_{u \in N(v) \cup \{v\}} W_l^u h_u^{(l)}), \forall l \in \{0, ..., L - 1\}$
    $h_v^{(l+1)} = \sigma(\sum_{u \in N(v)} W_l^u h_u^{(l)} + B_l h_v^{(l)}), \forall l \in \{0, ..., L - 1\}$
**GNN vs. CNN**

Convolutional neural network (CNN) layer with 3x3 filter:

GNN formulation: \( h^{(l+1)}_v = \sigma(W_l \sum_{u \in N(v)} \frac{h^{(l)}_u}{|N(v)|} + B_l h^{(l)}_v), \forall l \in \{0, \ldots, L - 1\} \)

CNN formulation: \( h^{(l+1)}_v = \sigma(\sum_{u \in N(v)} W^u_l h^{(l)}_u + B_l h^{(l)}_v), \forall l \in \{0, \ldots, L - 1\} \)

**Key difference:** We can learn different \( W^u_l \) for different “neighbor” \( u \) for pixel \( v \) on the image. The reason is we can pick an order for the 9 neighbors using relative position to the center pixel: \{(-1,-1), (-1,0), (-1, 1), ..., (1, 1)\}
Convolutional neural network (CNN) layer with 3x3 filter:

- CNN can be seen as a special GNN with fixed neighbor size and ordering:
  - The size of the filter is pre-defined for a CNN.
  - The advantage of GNN is it processes arbitrary graphs with different degrees for each node.

Key difference: We can learn different $W^u$ for different “neighbor” $u$ for pixel $v$ on the image. The reason is we can pick an order for the 9 neighbors using relative position to the center pixel: \{(-1,-1), (-1,0), (-1, 1), …, (1, 1)\}
GNN vs. CNN

Convolutional neural network (CNN) layer with 3x3 filter:

- CNN can be seen as a special GNN with fixed neighbor size and ordering.
- CNN is not permutation equivariant.
- Switching the order of pixels will lead to different outputs.

**Key difference:** We can learn different $W^u_i$ for different “neighbor” $u$ for pixel $v$ on the image. The reason is we can pick an order for the 9 neighbors using relative position to the center pixel: {(-1,-1), (-1,0), (-1, 1), ..., (1, 1)}
Transformer is one of the most popular architectures that achieves great performance in many sequence modeling tasks.

**Key component: self-attention**
- Every token/word attends to all the other tokens/words via matrix calculation.
Transformer layer can be seen as a special GNN that runs on a fully-connected “word” graph!

Since each word attends to all the other words, the computation graph of a transformer layer is identical to that of a GNN on the fully-connected “word” graph.
Stanford CS224W: A General Perspective on Graph Neural Networks
A General GNN Framework (1)

GNN Layer = Message + Aggregation

- Different instantiations under this perspective
- GCN, GraphSAGE, GAT, ...

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020
A General GNN Framework (2)

Connect GNN layers into a GNN
- Stack layers sequentially
- Ways of adding skip connections

(3) Layer connectivity

J. You, R. Ying, J. Leskovec. *Design Space of Graph Neural Networks*, NeurIPS 2020
A General GNN Framework (3)

Idea: Raw input graph ≠ computational graph
- Graph feature augmentation
- Graph structure augmentation

(4) Graph augmentation
How do we train a GNN

- Supervised/Unsupervised objectives
- Node/Edge/Graph level objectives

(We will discuss all of these later in class)
GNN Framework: Summary

(1) Message

(2) Aggregation

(3) Layer connectivity

(4) Graph augmentation

(5) Learning objective

TARGET NODE

INPUT GRAPH

D
E
B
A
C

GNN Layer 1

GNN Layer 2

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020
Stanford CS224W: A Single Layer of a GNN

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

GNN Layer = Message + Aggregation

• Different instantiations under this perspective
• GCN, GraphSAGE, GAT, ...

GNN Layer 1

(1) Message

(2) Aggregation

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020
## Idea of a GNN Layer:

- Compress a set of vectors into a single vector
- **Two-step process:**
  - (1) Message
  - (2) Aggregation

---

Node $v$:

- **(1) Message**
- **(2) Aggregation**

### GNN Layer

- **Input node embedding $h_v^{(l-1)}$, $h_u^{(l-1)}_{u \in N(v)}$**
- **Output node embedding $h_v^{(l)}$**

- **$l$-th GNN Layer**
(1) Message computation

Message function: \( m_u^{(l)} = MSG^{(l)}(h_u^{(l-1)}) \)

- **Intuition:** Each node will create a message, which will be sent to other nodes later

- **Example:** A Linear layer \( m_u^{(l)} = W^{(l)}h_u^{(l-1)} \)
  - Multiply node features with weight matrix \( W^{(l)} \)

(2) Aggregation

TARGET NODE

Node \( \nu \)

INPUT GRAPH

(1) Message
(2) Aggregation

**Intuition:** Each node will aggregate the messages from node \( v \)'s neighbors

\[ h_v^{(l)} = AGG^{(l)} \left( \left\{ m_u^{(l)}, u \in N(v) \right\} \right) \]

**Example:** \( \text{Sum}(\cdot), \text{Mean}(\cdot) \) or \( \text{Max}(\cdot) \) aggregator

\[ h_v^{(l)} = \text{Sum}(\{ m_u^{(l)}, u \in N(v) \}) \]
Message Aggregation: Issue

- **Issue:** Information from node \( v \) itself **could get lost**
  - Computation of \( h_v^{(l)} \) does not directly depend on \( h_v^{(l-1)} \)
- **Solution:** Include \( h_v^{(l-1)} \) when computing \( h_v^{(l)} \)
  - (1) **Message:** compute message from node \( v \) itself
    - Usually, a **different message computation** will be performed
      \[
      m_u^{(l)} = W^{(l)} h_u^{(l-1)} \quad m_v^{(l)} = B^{(l)} h_v^{(l-1)}
      \]
  - (2) **Aggregation:** After aggregating from neighbors, we can aggregate the message from node \( v \) itself
    - Via **concatenation** or **summation**
A Single GNN Layer

- Putting things together:
  - **(1) Message**: each node computes a message
    \[ m_u^{(l)} = \text{MSG}^{(l)} \left( h_u^{(l-1)} \right), u \in \{N(v) \cup v\} \]
  - **(2) Aggregation**: aggregate messages from neighbors
    \[ h_v^{(l)} = \text{AGG}^{(l)} \left( \{ m_u^{(l)}, u \in N(v) \} , m_v^{(l)} \right) \]
  - **Nonlinearity (activation)**: Adds expressiveness
    - Often written as \( \sigma(\cdot) \): ReLU(\cdot), Sigmoid(\cdot), ...
    - Can be added to message or aggregation
Classical GNN Layers: GCN (1)

- (1) Graph Convolutional Networks (GCN)

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

- How to write this as Message + Aggregation?

\[ h^{(l)}_v = \sigma \left( \sum_{u \in N(v)} W^{(l)} \frac{h^{(l-1)}_u}{|N(v)|} \right) \]
### Classical GNN Layers: GCN (2)

- **(1) Graph Convolutional Networks (GCN)**

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

- **Message:**
  - Each Neighbor: \( m_u^{(l)} = \frac{1}{|N(v)|} W^{(l)} h_u^{(l-1)} \)

- **Aggregation:**
  - Sum over messages from neighbors, then apply activation
  - \( h_v^{(l)} = \sigma \left( \text{Sum} \left( \{m_u^{(l)}, u \in N(v)\} \right) \right) \)

**Normalized by node degree**

(In the GCN paper they use a slightly different normalization)

In GCN graph is assumed to have self-edges that are included in the summation.
(2) GraphSAGE

\[ h_v^{(l)} = \sigma \left( W^{(l)} \cdot \text{CONCAT} \left( h_v^{(l-1)}, \text{AGG} \left( \{ h_u^{(l-1)}, \forall u \in N(v) \} \right) \right) \right) \]

- **How to write this as Message + Aggregation?**
  - **Message** is computed within the \( \text{AGG}(\cdot) \)
  - **Two-stage aggregation**
    - **Stage 1:** Aggregate from node neighbors
      \[ h_{N(v)}^{(l)} \leftarrow \text{AGG} \left( \{ h_u^{(l-1)}, \forall u \in N(v) \} \right) \]
    - **Stage 2:** Further aggregate over the node itself
      \[ h_v^{(l)} \leftarrow \sigma \left( W^{(l)} \cdot \text{CONCAT}(h_v^{(l-1)}, h_{N(v)}^{(l)}) \right) \]
GraphSAGE Neighbor Aggregation

- **Mean:** Take a weighted average of neighbors
  \[
  AGG = \frac{\sum_{u \in N(v)} h_u^{(l-1)}}{|N(v)|}
  \]

- **Pool:** Transform neighbor vectors and apply symmetric vector function Mean(·) or Max(·)
  \[
  AGG = \text{Mean}(\{\text{MLP}(h_u^{(l-1)}), \forall u \in N(v)\})
  \]

- **LSTM:** Apply LSTM to reshuffled of neighbors
  \[
  AGG = \text{LSTM}([h_u^{(l-1)}, \forall u \in \pi(N(v))])
  \]
GraphSAGE: L₂ Normalization

- **L₂ Normalization:**
  - Optional: Apply L₂ normalization to $h_v^{(l)}$ at every layer
  
  $h_v^{(l)} \leftarrow \frac{h_v^{(l)}}{\|h_v^{(l)}\|_2}$ \quad \forall v \in V$ where $\|u\|_2 = \sqrt{\sum_i u_i^2}$ (L₂-norm)

- Without L₂ normalization, the embedding vectors have different scales (L₂-norm) for vectors
- In some cases (not always), normalization of embedding results in performance improvement
- After L₂ normalization, all vectors will have the same L₂-norm
(3) Graph Attention Networks

\[ h_v^{(l)} = \sigma \left( \sum_{u \in N(v)} \alpha_{vu} W^{(l)} h_u^{(l-1)} \right) \]

In GCN / GraphSAGE

- \( \alpha_{vu} = \frac{1}{|N(v)|} \) is the \textbf{weighting factor (importance)} of node \( u \)'s message to node \( v \)
- \( \Rightarrow \alpha_{vu} \) is defined \textit{explicitly} based on the structural properties of the graph (node degree)
- \( \Rightarrow \) All neighbors \( u \in N(v) \) are equally important to node \( v \)
(3) Graph Attention Networks

$$h_v^{(l)} = \sigma(\sum_{u \in N(v)} \alpha_{vu} W^{(l)} h_u^{(l-1)})$$

Not all node’s neighbors are equally important

- **Attention** is inspired by cognitive attention.
- The **attention** $\alpha_{vu}$ focuses on the important parts of the input data and fades out the rest.
  - **Idea**: the NN should devote more computing power on that small but important part of the data.
  - Which part of the data is more important depends on the context and is learned through training.
Can we do better than simple neighborhood aggregation?

Can we let weighting factors $\alpha_{vu}$ to be learned?

- **Goal:** Specify **arbitrary importance** to different neighbors of each node in the graph.

- **Idea:** Compute embedding $h^{(l)}_v$ of each node in the graph following an **attention strategy**:
  - Nodes attend over their neighborhoods’ message.
  - Implicitly specifying different weights to different nodes in a neighborhood.

[Velickovic et al., ICLR 2018; Vaswani et al., NIPS 2017]
Let $\alpha_{vu}$ be computed as a byproduct of an attention mechanism $a$:

1. Let $a$ compute **attention coefficients** $e_{vu}$ across pairs of nodes $u$, $v$ based on their messages:

\[ e_{vu} = a(W^{(l)}h_u^{(l-1)}, W^{(l)}h_v^{(l-1)}) \]

- $e_{vu}$ indicates the importance of $u$'s message to node $v$
Attention Mechanism (2)

- **Normalize** $e_{vu}$ into the **final attention weight** $\alpha_{vu}$
  - Use the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:
    $$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

- **Weighted sum** based on the **final attention weight** $\alpha_{vu}$
  $$h_v^{(l)} = \sigma(\sum_{u \in N(v)} \alpha_{vu} W^{(l)} h_u^{(l-1)})$$

  **Weighted sum using** $\alpha_{AB}$, $\alpha_{AC}$, $\alpha_{AD}$:
  $$h_A^{(l)} = \sigma(\alpha_{AB} W^{(l)} h_B^{(l-1)} + \alpha_{AC} W^{(l)} h_C^{(l-1)} + \alpha_{AD} W^{(l)} h_D^{(l-1)})$$

What is the form of attention mechanism $a$?

The approach is agnostic to the choice of $a$

- E.g., use a simple single-layer neural network
  - $a$ have trainable parameters (weights in the Linear layer)

Parameters of $a$ are trained jointly:

- Learn the parameters together with weight matrices (i.e., other parameter of the neural net $W^{(l)}$) in an end-to-end fashion
**Attention Mechanism (4)**

- **Multi-head attention**: Stabilizes the learning process of attention mechanism
  - Create multiple attention scores (each replica with a different set of parameters):
    - $h_v^{(l)}[1] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^1 W^{(l)} h_u^{(l-1)})$
    - $h_v^{(l)}[2] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^2 W^{(l)} h_u^{(l-1)})$
    - $h_v^{(l)}[3] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^3 W^{(l)} h_u^{(l-1)})$
  - Outputs are aggregated:
    - By concatenation or summation
    - $h_v^{(l)} = AGG(h_v^{(l)}[1], h_v^{(l)}[2], h_v^{(l)}[3])$
Benefits of Attention Mechanism

- **Key benefit:** Allows for (implicitly) specifying different importance values $(\alpha_{vu})$ to different neighbors

- **Computationally efficient:**
  - Computation of attentional coefficients can be parallelized across all edges of the graph
  - Aggregation may be parallelized across all nodes

- **Storage efficient:**
  - Sparse matrix operations do not require more than $O(V + E)$ entries to be stored
  - **Fixed** number of parameters, irrespective of graph size

- **Localized:**
  - Only attends over local network neighborhoods

- **Inductive capability:**
  - It is a shared *edge-wise* mechanism
  - It does not depend on the global graph structure
Stanford CS224W: GNN Layers in Practice
In practice, these classic GNN layers are a great starting point.

- We can often get better performance by considering a general GNN layer design.
- Concretely, we can include modern deep learning modules that proved to be useful in many domains.

A suggested GNN Layer:
- Linear
- BatchNorm
- Dropout
- Activation
- Attention
- Aggregation
- Transformation
Many modern deep learning modules can be incorporated into a GNN layer

- **Batch Normalization:**
  - Stabilize neural network training

- **Dropout:**
  - Prevent overfitting

- **Attention/Gating:**
  - Control the importance of a message

- **More:**
  - Any other useful deep learning modules
Batch Normalization

- **Goal**: Stabilize neural networks training
- **Idea**: Given a batch of inputs (node embeddings)
  - Re-center the node embeddings into zero mean
  - Re-scale the variance into unit variance

**Input**: \( X \in \mathbb{R}^{N \times D} \)
\( N \) node embeddings

**Trainable Parameters**: \( \gamma, \beta \in \mathbb{R}^D \)

**Output**: \( Y \in \mathbb{R}^{N \times D} \)
Normalized node embeddings

**Step 1:**
Compute the mean and variance over \( N \) embeddings

\[
\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j} \\
\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2
\]

**Step 2:**
Normalize the feature using computed mean and variance

\[
\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \\
y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j
\]

**Goal**: Regularize a neural net to prevent overfitting.

**Idea**:

- **During training**: with some probability $p$, randomly set neurons to zero (turn off)
- **During testing**: Use all the neurons for computation

---

Srivastava et al. [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](http://jmlr.org/papers/v15/srivastava14a.html), JMLR 2014

---

10/12/21
In GNN, Dropout is applied to the linear layer in the message function.

A simple message function with linear layer:

$$m_u^{(l)} = W^{(l)}h_u^{(l-1)}$$
Activation (Non-linearity)

Apply activation to $i$-th dimension of embedding $x$

- **Rectified linear unit (ReLU)**
  \[
  \text{ReLU}(x_i) = \max(x_i, 0)
  \]
  - Most commonly used

- **Sigmoid**
  \[
  \sigma(x_i) = \frac{1}{1 + e^{-x_i}}
  \]
  - Used only when you want to restrict the range of your embeddings

- **Parametric ReLU**
  \[
  \text{PReLU}(x_i) = \max(x_i, 0) + a_i \min(x_i, 0)
  \]
  - $a_i$ is a trainable parameter
  - Empirically performs better than ReLU
Summary: Modern deep learning modules can be included into a GNN layer for better performance.

Designing novel GNN layers is still an active research frontier!

Suggested resources: You can explore diverse GNN designs or try out your own ideas in GraphGym.
Stanford CS224W: Stacking Layers of a GNN
How to connect GNN layers into a GNN?

• Stack layers sequentially
• Ways of adding skip connections

(3) Layer connectivity

GNN Layer 1

GNN Layer 2

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020
Stacking GNN Layers

How to construct a Graph Neural Network?

- The standard way: Stack GNN layers sequentially
- Input: Initial raw node feature \( \mathbf{x}_v \)
- Output: Node embeddings \( \mathbf{h}_v^{(L)} \) after \( L \) GNN layers

\[
\mathbf{h}_v^{(0)} = \mathbf{x}_v
\]
The Over-smoothing Problem

- **The Issue of stacking many GNN layers**
  - GNN suffers from the over-smoothing problem
- **The over-smoothing problem:** all the node embeddings converge to the same value
  - This is bad because we want to use node embeddings to differentiate nodes
- Why does the over-smoothing problem happen?
Receptive Field of a GNN

**Receptive field:** the set of nodes that determine the embedding of a node of interest

- In a $K$-layer GNN, each node has a receptive field of $K$-hop neighborhood

![Receptive field for 1-layer GNN](image1)

![Receptive field for 2-layer GNN](image2)

![Receptive field for 3-layer GNN](image3)
Receptive Field of a GNN

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

1-hop neighbor overlap
Only 1 node

2-hop neighbor overlap
About 20 nodes

3-hop neighbor overlap
Almost all the nodes!
We can explain over-smoothing via the notion of receptive field

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

Next: how do we overcome over-smoothing problem?
What do we learn from the over-smoothing problem?

**Lesson 1: Be cautious when adding GNN layers**

- Unlike neural networks in other domains (CNN for image classification), adding more GNN layers do not always help.

**Step 1:** Analyze the necessary receptive field to solve your problem. E.g., by computing the diameter of the graph.

**Step 2:** Set number of GNN layers $L$ to be a bit more than the receptive field we like. **Do not set $L$ to be unnecessarily large!**

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

**Solution 1:** Increase the expressive power within each GNN layer

- In our previous examples, each transformation or aggregation function only include one linear layer
- We can make aggregation / transformation become a deep neural network!

If needed, each box could include a 3-layer MLP
How to make a shallow GNN more expressive?

**Solution 2:** Add layers that do not pass messages

- A GNN does not necessarily only contain GNN layers
  - E.g., we can add **MLP layers** (applied to each node) before and after GNN layers, as **pre-process layers** and **post-process layers**

**Pre-processing layers:** Important when encoding node features is necessary. E.g., when nodes represent images/text

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

In practice, adding these layers works great!
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: $F(x)$

After adding shortcuts: $F(x) + x$
Why do skip connections work?

- **Intuition:** Skip connections create a mixture of models
- $N$ skip connections $\rightarrow 2^N$ possible paths
- Each path could have up to $N$ modules
- We automatically get a mixture of shallow GNNs and deep GNNs

Path 1: include this module
(a) Conventional 3-block residual network

Path 2: skip this module
(b) Unraveled view of (a)

All the possible paths: $2 \times 2 \times 2 = 2^3 = 8$
Example: GCN with Skip Connections

- **A standard GCN layer**

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

  This is our \( F(x) \)

- **A GCN layer with skip connection**

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

  \[ F(x) + x \]
Other Options of Skip Connections

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