Contents

Overview

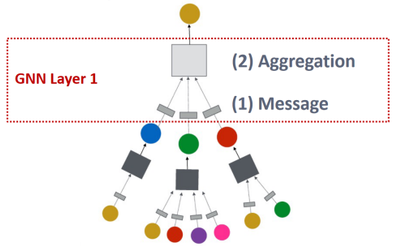
A general perspective on GNN

General GNN framework

1, 2: Message, aggregation

GNN Layer = Message + Aggregation

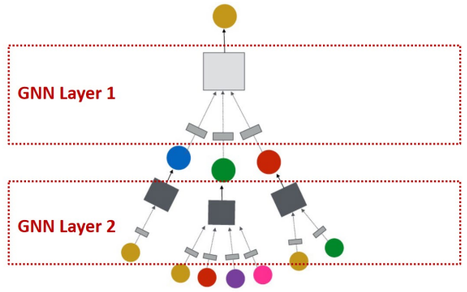
- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**



3: Layer connectivity

Connect GNN layers into a GNN

- Stack layers sequentially
- Ways of adding skip connections



4: Graph augmentation

Idea: Raw input graph ≠ computational graph

- Graph feature augmentation
- Graph structure augmentation

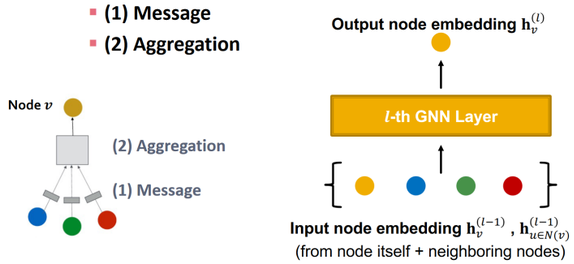
5: Learning objective

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

One GNN layer

- The aggregation needs to be **order invariant**

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



Message computation

- **Message function:** $\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$
  - **Intuition:** Each node will create a message, which will be sent to other nodes later
  - **Example:** A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$
    - Multiply node features with weight matrix $\mathbf{W}^{(l)}$

Issue & solutions – itself message lost

- **Message.** Message from the node itself has different message computation compared to its neighbours. Denoted by B compared to W.
- **Aggregations.** Aggregate itself embedding & concat with the neighbour aggregation

- **Solution:** Include $\mathbf{h}_v^{(l-1)}$ when computing $\mathbf{h}_v^{(l)}$
  - **(1) Message: compute message from node v itself**
    - Usually, a **different message computation** will be performed

      $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$ $\mathbf{m}_v^{(l)} = \mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)}$
  - **(2) Aggregation:** After aggregating from neighbors, we can **aggregate the message from node v itself**
    - Via **concatenation** or **summation**

Then aggregate from node itself

$$\mathbf{h}_v^{(l)} = \text{CONCAT}\left(\text{AGG}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right), \mathbf{m}_v^{(l)}\right)$$

First aggregate from neighbors

GCN baesd Summary

- Message is some transformation on the embedding
- Aggregate the message from the neighbour and itself

  - **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 $\sigma(\cdot)$: ReLU$(\cdot)$, Sigmoid$(\cdot)$, ...
      - Can be added to **message or aggregation**

GCN as message transformation + aggregation

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

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

Message transformation + aggregation.

Message

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

Aggregation

Details: normalise by node degree & sum

- **Message:**
  - Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$ Normalized by node degree (In the GCN paper they use a slightly different normalization)
- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\text{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$ In GCN graph is assumed to have self-edges that are included in the summation.

GraphSAGE

Different to GCN

- The aggregation has many choices
- Aggregation is baesd on the embedding over the computated message (some linear transformation of the embedding)

  - **(2) GraphSAGE**

    $\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(l-1)}, \text{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)\right)\right)$
  - **How to write this as Message + Aggregation?**
    - **Message** is computed within the $\text{AGG}(\cdot)$
    - **Two-stage aggregation**
      - **Stage 1:** Aggregate from node neighbors

        $\mathbf{h}_{N(v)}^{(l)} \leftarrow \text{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)$
      - **Stage 2:** Further aggregate over the node itself

        $\mathbf{h}_v^{(l)} \leftarrow \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)})\right)$

Neighbour aggregation – mean, pool (MLP), LSTM

- LSTM is not **ordering invariance**
  - Need to permute the order

  - **Mean:** Take a weighted average of neighbors

    $\text{AGG} = \sum_{u \in N(v)}\frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}$ Aggregation / Message computation
  - **Pool:** Transform neighbor vectors and apply symmetric vector function Mean$(\cdot)$ or Max$(\cdot)$

    $\text{AGG} = \text{Mean}(\{\text{MLP}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$

    Aggregation / Message computation
  - **LSTM:** Apply LSTM to reshuffled of neighbors

    $\text{AGG} = \text{LSTM}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))])$

    Aggregation

L2 Normalisation – layer norm 4 embedding

- Without [0, 1] normalisation, the embedding has different scale

  - **Optional:** Apply ℓ_2 normalization to $\mathbf{h}_v^{(l)}$ at every layer
  - $\mathbf{h}_v^{(l)} \leftarrow \frac{\mathbf{h}_v^{(l)}}{\|\mathbf{h}_v^{(l)}\|_2}$ $\forall v \in V$ where $\|u\|_2 = \sqrt{\sum_i u_i^2}$ (ℓ_2-norm)
  - Without ℓ_2 normalization, the embedding vectors have different scales (ℓ_2-norm) for vectors
  - In some cases (not always), normalization of embedding results in performance improvement
  - After ℓ_2 normalization, all vectors will have the same ℓ_2-norm

GAT

- Attention weight α_{vu} – the importance of a message coming from the node u (neighbour) for node v who is aggregating the message
- For GCN / GraphSage – neighbours of same importance to node v
- Initution – allow the model to learn which part is important
  - **(3) Graph Attention Networks**

    $\mathbf{h}_v^{(l)} = \sigma(\sum_{u \in N(v)}\alpha_{vu}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)})$

    Attention weights
  - **In GCN / GraphSAGE**
    - $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u's message to node v
    - $\Longrightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph (node degree)
    - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

Not all node are important

- **Attention** is inspired by cognitive attention.
- The **attention α_{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.

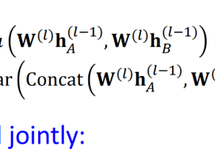
- **Goal:** Specify **arbitrary importance** to different neighbors of each node in the graph
- **Idea:** Compute embedding $\boldsymbol{h}_v^{(l)}$ of each node in the graph following an **attention strategy**:
  - Nodes attend over their neighborhoods' message
  - Implicitly specifying different weights to different nodes in a neighborhood

Attention coefficient

- Some function based on the previous layer embedding of the related two nodes
  - Let α_{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:

$$e_{vu} = a(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)}\boldsymbol{h}_v^{(l-1)})$$

    - **e_{vu} indicates the importance of u's message to node v**



$$e_{AB} = a(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)})$$

Attention weighted sum aggregation; softmax [0,1]

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

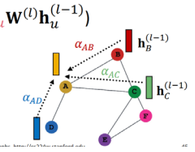
$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

- **Weighted sum** based on the **final attention weight α_{vu}**

$$\mathbf{h}_v^{(l)} = \sigma(\textstyle\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

Weighted sum using α_{AB}, α_{AC}, α_{AD}:

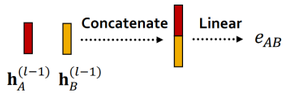
$$\mathbf{h}_A^{(l)} = \sigma(\alpha_{AB}\mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)} + \alpha_{AC}\mathbf{W}^{(l)}\mathbf{h}_C^{(l-1)} + \alpha_{AD}\mathbf{W}^{(l)}\mathbf{h}_D^{(l-1)})$$



Attention Form

- Learn jointly – attention weight & transform weight matrix

- **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)



$$e_{AB} = a\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)$$
$$= \text{Linear}\left(\text{Concat}\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)\right)$$

  - Parameters of a are trained jointly:
    - Learn the parameters together with weight matrices (i.e., other parameter of the neural net $\mathbf{W}^{(l)}$) in an end-to-end fashion

Multi-head attention

- Stabilizes the learning process of attention mechanism
- Only 1 attention score may stuck at local minimum
  - Multi-head attention then aggregate them
  - The learning more stabilse
- When use, use each separtely to calculate
- Weight initialisation should be different

  - **Create multiple attention scores** (each replica with a different set of parameters):

$$\mathbf{h}_v^{(l)}[1] = \sigma(\textstyle\sum_{u \in N(v)} \alpha_{vu}^1 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$
$$\mathbf{h}_v^{(l)}[2] = \sigma(\textstyle\sum_{u \in N(v)} \alpha_{vu}^2 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$
$$\mathbf{h}_v^{(l)}[3] = \sigma(\textstyle\sum_{u \in N(v)} \alpha_{vu}^3 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

  - **Outputs are aggregated:**
    - By concatenation or summation
    - $\mathbf{h}_v^{(l)} = \text{AGG}(\mathbf{h}_v^{(l)}[1], \mathbf{h}_v^{(l)}[2], \mathbf{h}_v^{(l)}[3])$

Benefit

- **Key benefit:** Allows for (implicitly) specifying **different importance values (α_{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

In practice

Batch Normalization:

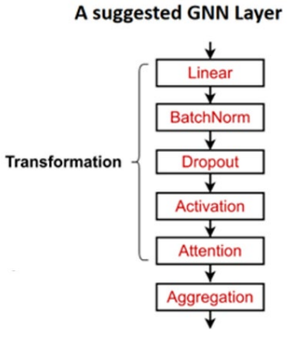
Stabilize neural network training

Dropout:

Prevent overfitting

Attention/Gating:

Control the importance of a message



Batch normalisation

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

Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$
N node embeddings

Trainable Parameters:
$\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^D$

Output: $\mathbf{Y} \in \mathbb{R}^{N \times D}$
Normalized node embeddings

Step 1:
Compute the mean and variance over N embeddings

$$\boldsymbol{\mu}_j = \frac{1}{N}\sum_{i=1}^{N} \mathbf{X}_{i,j}$$
$$\boldsymbol{\sigma}_j^2 = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{X}_{i,j} - \boldsymbol{\mu}_j)^2$$

Step 2:
Normalize the feature using computed mean and variance

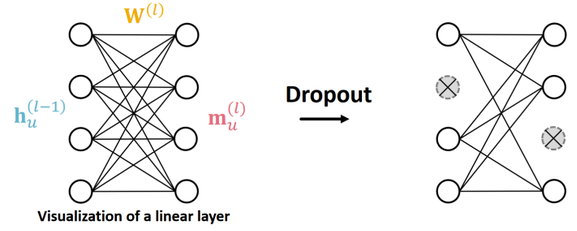
$$\widehat{\mathbf{X}}_{i,j} = \frac{\mathbf{X}_{i,j} - \boldsymbol{\mu}_j}{\sqrt{\boldsymbol{\sigma}_j^2 + \epsilon}}$$
$$\mathbf{Y}_{i,j} = \boldsymbol{\gamma}_j \widehat{\mathbf{X}}_{i,j} + \boldsymbol{\beta}_j$$

Dropout

- **Goal**: Regularize a neural net to prevent overfitting
- During **training**: with some probability p, randomly set neurons to zero (turn off)
- During **testing**: Use all the neurons for computation

- In GNN, Dropout is applied to the linear layer in the message function

  - **A simple message function with linear layer: $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$**

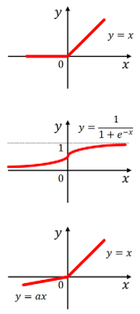


Visualization of a linear layer

Activation (Non-linearity)

Apply activation to i-th dimension of embedding x

- **Rectified linear unit (ReLU)**
  $\text{ReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0)$
  - Most commonly used
- **Sigmoid**
  $\sigma(\mathbf{x}_i) = \frac{1}{1+e^{-\mathbf{x}_i}}$
  - Used only when you want to restrict the range of your embeddings
- **Parametric ReLU**
  $\text{PReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0) + a_i \min(\mathbf{x}_i, 0)$
  a_i is a trainable parameter
  - Empirically performs better than ReLU



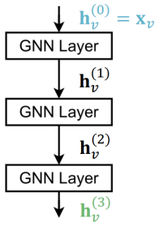
Stack GNN layers

How to connect GNN layers into a GNN?

- Stack layers **sequentially**
- Ways of adding **skip connections**

Sequentially

- **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



The depth of GNN

- Tells how many hops to aggregate
- Different to CNN
  - Not about the expressiveness of the model

Over-smoothing problem

- Issue of **stacking many** GNN layers
- all the node embeddings converge to the **same value**
  - all the node embeddings converge to the same value
  - bad because we want to use node embeddings to differentiate nodes
- **reason.**
  - The receptive field is too large and so the info collected for each node are similar

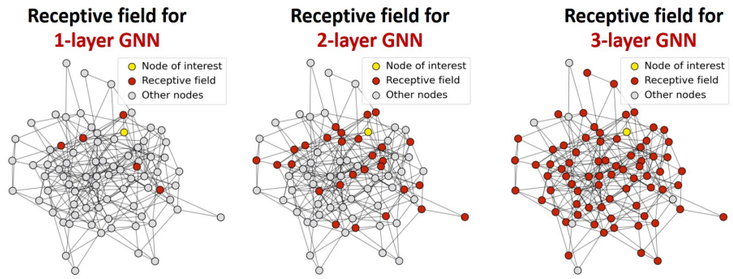
Receptive field

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

2 layers

- Neighbour of neighbours
  - 1 hop or 2 hops neighbourhood
- The shared neighbours (in red) between the two yellow nodes quickly grows when we increase the number of hops (num of GNN layers)



Receptive field & over-smoothing

- 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

1. Stack many GNN layers
2. nodes will have highly overlapped receptive fields
3. node embeddings will be highly similar
4. suffer from the oversmoothing problem

Solutions

get what's needed

- 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
  - #GNN > receptive field

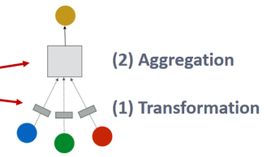
Shallow GNN more expressive

- Expressive power for one layer

1 - NN as aggregation

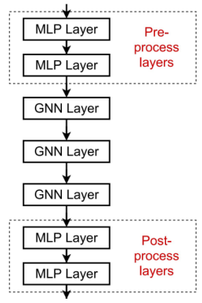
- We can **make aggregation / transformation become a deep neural network**!

If needed, each box could include a 3-layer MLP



2 - Combine classic NN with GNN

- 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

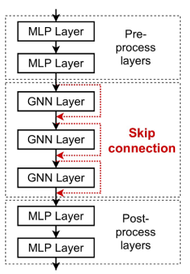
Post-processing layers: Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

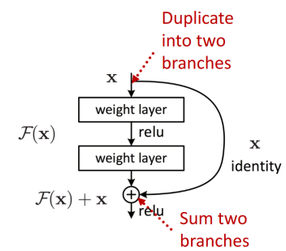
In practice, adding these layers works great!

Need many layers

1 - Add skip connections in GNNs

- Observation from over-smoothing: Node embeddings in **earlier GNN layers** can sometimes better differentiate node
- **Solution**: We can increase the impact of earlier layers on the final node embeddings, by adding shortcuts in GNN

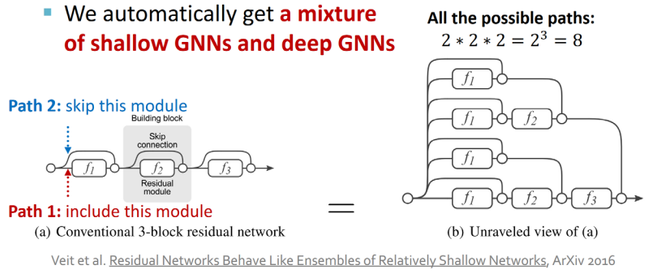




Idea of skip connections:
Before adding shortcuts:
$\boldsymbol{F(x)}$
After adding shortcuts:
$\boldsymbol{F(x) + x}$

Idea of skip connection – mixture of models

- **Why do skip connections work?**
  - **Intuition:** Skip connections create **a mixture of models**
  - N skip connections → 2^N possible paths
  - Each path could have up to N modules
  - We automatically get **a mixture of shallow GNNs and deep GNNs**

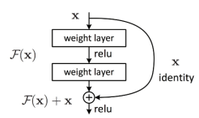


Veit et al. Residual Networks Behave Like Ensembles of Relatively Shallow Networks, ArXiv 2016

GCN with skip connection

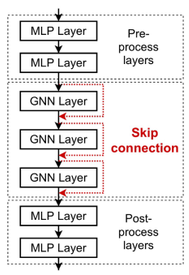
- **A standard GCN layer**
- $\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$

This is our $F(\mathbf{x})$



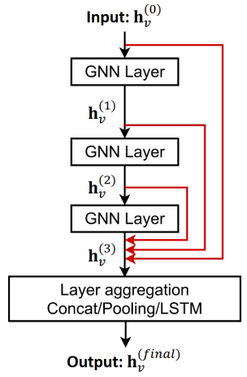
- **A GCN layer with skip connection**
- $\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} + \mathbf{h}_v^{(l-1)}\right)$

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



2- Directly skip to the last layer

- The final layer directly aggregates from the all the node embeddings in the previous layers
  - E.g. add info from early layer e.g. 1-hop neighbour or deeper ones e.g. 4-hop neighbour
- Xu et al. Representation learning on graphs with jumping knowledge networks, ICML 2018



Summary

Input node embedding

$$h_v^{(l-1)}, h_{u \in N(v)}^{(l-1)}$$

Message

A linear layer $\quad m_u^{(l)} = W^{(l)} h_u^{(l-1)},$

Message transformation $\quad m_u^{(l)} = MSG^{(l)}\left(h_u^{(l-1)}\right)$

Aggregation

$$h_v^{(l)} = \text{CONCAT}\left(\text{AGG}\left(\left\{m_u^{(l)}, u \in N(v)\right\}\right) m_v^{(l)i}\right)$$

GCN

Embedding

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

Message

$$m_u^{(l)} = \frac{1}{|N(v)|} W^{(l)} h_u^{(l-1)}$$

Aggregation

$$h_v^{(l)} = \sigma\left(Sum\left(\left\{m_u^{(l)}, u \in N(v)\right\}\right)\right)$$

GraphSage

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(l-1)}, \text{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)\right)\right)$$

Stage 1: Aggregate from node neighbors

$$\mathbf{h}_{N(v)}^{(l)} \leftarrow \text{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)$$

Stage 2: Further aggregate over the node itself

$$\mathbf{h}_v^{(l)} \leftarrow \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)})\right)$$

AGG

- **Mean:** Take a weighted average of neighbors

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}$$

Aggregation — Message computation

- **Pool:** Transform neighbor vectors and apply symmetric vector function Mean(·) or Max(·)

$$\text{AGG} = \text{Mean}(\{\text{MLP}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$$

Aggregation — Message computation

- **LSTM:** Apply LSTM to reshuffled of neighbors

$$\text{AGG} = \text{LSTM}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))])$$

Aggregation

GAT

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Attention weights

Attention coefficient

- Multi-head
  - Instead of one, use many independent heads to get differnet attention coefficient measuruign the importances of neighbours to the target node to aovid local minimum

$$e_{vu} = a(\mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)} \mathbf{h}_v^{(l-1)})$$

Over-smoothing problems

- collect too many information from neighbours and if all nodes do so, then very similar
  - Too many layers
  - layer in GNN refers to receptive field i.e. neighbour of a node (info to be collected)
- E.g. if receptive field is large enough to cover the whole graph

Receptive field & over-smoothing

- Treat increase in #GNN as an increase in receptive field

Solution – increase expressive power of GNN

1. NN as aggregation
2. Skip connection
   a. Skip to last layer