The Graph Neural Network Model

Presenter: Jiwon Jeong
jjwon4086@korea.ac.kr

Agenda

- 5. The Graph Neural Network Model
  - 5.1 Neural Message Passing
  - 5.2 Generalized Neighborhood Aggregation
  - 5.3 Generalized Update Methods
  - 5.4 Edge Features and Multi-relational GNNs
  - 5.5 Graph Pooling
  - 5.6 Generalized Message Passing

The Graph Neural Network Model

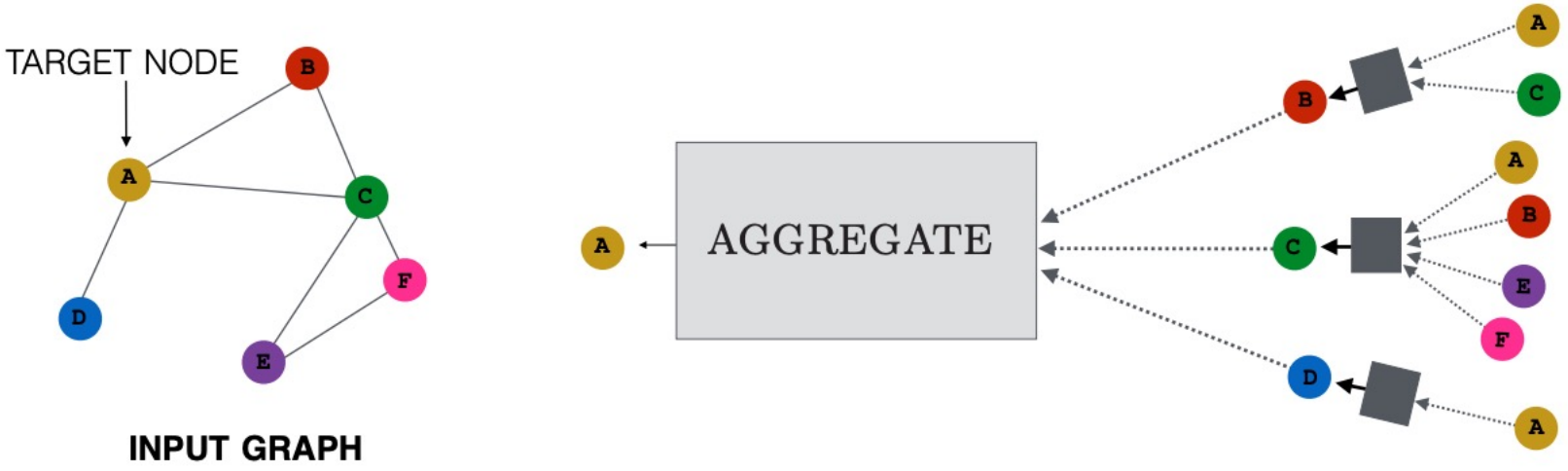
- Background
  - grid-structured: CNNs
  - sequences: RNNs
  - graph-structured: need to define new architectures!
- Graph Neural Network (GNN)
  - general framework for defining deep neural networks on graph data

Neural Message Passing

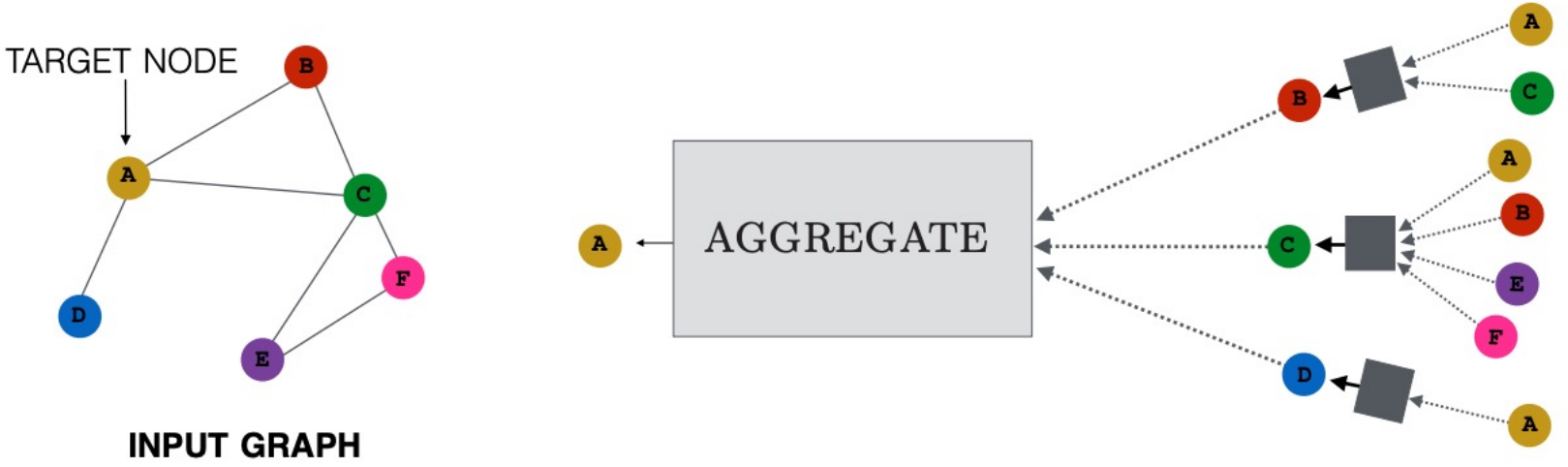
- Feature of GNN: a form of neural message passing
  - how we can take an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
  - set of node features $\boldsymbol{X} \in \mathbb{R}^{d \times |\mathcal{V}|}$
  - node embeddings $\boldsymbol{z}_u, \forall u \in \mathcal{V}$
  - embeddings for subgraphs and entire graphs

Neural Message Passing

- During each message-passing iteration in a GNN,
  a hidden embedding $\boldsymbol{h}_u^{(k)}$ corresponding to each node $u \in \mathcal{V}$ is updated according to information aggregated from u's graph neighborhood $\mathcal{N}(u)$.



Neural Message Passing



$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, v \in \mathcal{N}(u)\})\right)$$
$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right)$$

Neural Message Passing

- Formula

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, v \in \mathcal{N}(u)\})\right)$$
$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right)$$

  - AGGREGATE generates a message $\boldsymbol{m}_{\mathcal{N}(u)}^{(k)}$ based on aggregated neighborhood information
  - UPDATE combines the message $\boldsymbol{m}_{\mathcal{N}(u)}^{(k)}$ with the previous embedding $\boldsymbol{h}_u^{(k)}$ to generate the updated embedding $\boldsymbol{h}_u^{(k+1)}$
  - Initial embeddings: $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$
  - Output of the final layer: $\boldsymbol{z}_u = \boldsymbol{h}_u^{(K)}, \forall u \in \mathcal{V}$

Neural Message Passing

- Formula

$$\mathbf{h}_u^{(k+1)} = \mathrm{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathrm{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, v \in \mathcal{N}(u)\})\right)$$
$$= \mathrm{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right)$$

  - Since AGGREGATE function takes a *set* as input, GNNs are permutation equivariant by design

Permutation invariance and equivariance

- In a deep neural network over graph, any function f that takes and adjacency matrix A as input should ideally satisfy one of the two following properties:

$$f(\mathbf{PAP}^\top) = f(\mathbf{A}) \quad (\text{Permutation Invariance})$$
$$f(\mathbf{PAP}^\top) = \mathbf{P}f(\mathbf{A}) \quad (\text{Permutation Equivariance})$$

  where $\boldsymbol{P}$ is a permutation matrix.

- Permutation Invariance:
  the function does not depend on the arbitrary ordering of the row/columns in the adjacency matrix
- Permutation Equivariance:
  the output of f is permuted in a consistent way when we permute the adjacency matrix.

Node features

- In Part 1, we discussed shallow embedding methods
- Node features $\boldsymbol{x}_u, \forall u \in \mathcal{V}$ as input to the model

- No node features?
  - Node statistics: node degree, node centrality, … (in Section 2.1.)
  - Identity features: one-hot indicator feature
    - makes the model transductive and incapable of generalizing to unseen nodes

Neural Message Passing

- Motivations and Intuitions
  - After k iterations every node embedding contains information about its k-hop neighborhood
  - Information?
    - Structural information:
      the degrees of all the nodes in k-hop neighborhood
    - Feature-based:
      all the features in their k-hop neighborhood
      - this local feature-aggregation is analogous to the convolutional kernels (CNNs)
      - the connection between GNNs and convolutions in more detail in Chapter 7

Neural Message Passing

- The basic GNN message passing is defined as

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

  - where $\boldsymbol{W} \in \mathbb{R}^{d \times d}$ is trainable parameter matrix and σ denotes an elementwise non-linearity
  - the bias term $\boldsymbol{b} \in \mathbb{R}^{d}$ is often omitted for notational simplicity
  - it relies on linear operations followed by a single elementwise non-linearity -> MLP

Neural Message Passing

- Equivalently,

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v,$$

$$\text{UPDATE}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \sigma(\mathbf{W}_{\text{self}}\mathbf{h}_u + \mathbf{W}_{\text{neigh}}\mathbf{m}_{\mathcal{N}(u)})$$

  - where we recall that we use

$$\mathbf{m}_{\mathcal{N}(u)} = \text{AGGREGATE}^{(k)}(\{\mathbf{v}^{(k)}, \forall v \in \mathcal{N}(u)\})$$

Node vs. graph-level equations

- The graph-level definition of the model as follows:

$$\mathbf{H}^{(t)} = \sigma\left(\mathbf{A}\mathbf{H}^{(k-1)}\mathbf{W}_{\text{neigh}}^{(k)} + \mathbf{H}^{(k-1)}\mathbf{W}_{\text{self}}^{(k)}\right)$$

  - $\boldsymbol{H}^{(k)} \in \mathbb{R}^{|V| \times d}$ denotes the matrix of node representations at layer t
  - $\boldsymbol{A}$ is the graph adjacency matrix
  - we omitted the bias term

- The node-level definition:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

Neural Message Passing

- Self-loops

$$\mathbf{h}_u^{(k)} = \text{AGGREGATE}(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u) \cup \{u\}\})$$

  - No need to define an explicit update function
  - Alleviate overfitting
  - Limits the expressivity: cannot differentiate the information from neighbors or itself

- Graph-level

$$\mathbf{H}^{(t)} = \sigma\left((\mathbf{A} + \mathbf{I})\mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\right)$$

Generalized Methods

- The basic GNN can be improved upon and generalized!
- 5.2. Generalized Neighborhood Aggregation
  - how the AGGREGATE operator can be generalized and improved upon
- 5.3. Generalized Update Methods
  - how the UPDATE operator can be generalized and improved upon

Generalized Neighborhood Aggregation

- Neighborhood Normalization
  - The most basic neighborhood aggregation operation: Sum

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v$$

    - Unstable and highly sensitive to node degrees
    - If node u has 100x as many neighbors as node u', $\| \Sigma_{v \in \mathcal{N}(u)} \boldsymbol{h}_u \| \gg \| \Sigma_{v' \in \mathcal{N}(u')} \boldsymbol{h}_{u'} \|$
    - Lead to numerical instabilities and difficulties for optimization

Generalized Neighborhood Aggregation

- Neighborhood Normalization
  - Normalize the aggregation operation based upon the degrees of the nodes involved
  - Average

$$\mathbf{m}_{\mathcal{N}(u)} = \frac{\sum_{v \in \mathcal{N}(u)} \mathbf{h}_v}{|\mathcal{N}(u)|}$$

  - Symmetric normalization

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}$$

    - a first-order approximation of spectral graph convolution -> Chapter 7

Generalized Neighborhood Aggregation

- Graph convolution networks (GCNs)
  - the symmetric-normalized aggregation
  - self-loop update approach
  - message passing function:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}\right)$$

To normalize or not to normalize?

- Pros of normalization
  - stable and strong performance
  - helpful in tasks where node feature information is far more useful than structural information
    or a very wide range of node degrees can lead to instabilities
- Cons of normalization
  - lead to a loss of information
    - hard to distinguish between nodes of different degrees and various other structural graph features
- The use of normalization is an application-specific question!

Generalized Neighborhood Aggregation

- Improving the AGGREGATE operator
  - Since the neighborhood aggregation operation is a set function ($\{\mathbf{h}_v, \forall v \in \mathcal{N}(u)\} \xrightarrow{\text{maps}} \mathbf{m}_{\mathcal{N}(u)}$), any aggregation function must be permutation invariant
  - Pooling!

Generalized Neighborhood Aggregation

- Set pooling
  - universal set function approximator

$$\mathbf{m}_{\mathcal{N}(u)} = \mathrm{MLP}_\theta \left(\sum_{v \in \mathcal{N}(u)} \mathrm{MLP}_\phi(\mathbf{h}_v) \right)$$

    - possible to replace the sum with an alternative reduction function (element-wise maximum or minimum)
    - small increases in performance
    - increased risk of overfitting: careful to overparameterization

Generalized Neighborhood Aggregation

- Janossy pooling
  - Janossy pooling employs a different approach entirely
    : permutation-invariant reduction -> permutation-sensitive function and average the results
  - $\pi_i \in \Pi$ denotes a permutation function
    $\{\mathbf{h}_v, \forall v \in \mathcal{N}(u)\} \overset{\text{maps}}{\longrightarrow} (\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, ..., \mathbf{h}_{v_{|\mathcal{N}(v)|}})_{\pi_i}$
  - Aggregation function with Janossy pooling

$$\mathbf{m}_{(u)} = \text{MLP}_\theta \left(\frac{1}{|\Pi|} \sum_{\pi \in \Pi} \rho_\phi(\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, ..., \mathbf{h}_{v_{|\mathcal{N}(u)}})_{\pi_i} \right)$$

    - ρ_ϕ is a permutation-sensitive function -> usually LSTM

Generalized Neighborhood Aggregation

- Janossy pooling
  - Aggregation function with Janossy pooling

$$\mathbf{m}_{(u)} = \mathrm{MLP}_\theta \left(\frac{1}{|\Pi|} \sum_{\pi \in \Pi} \rho_\phi(\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, ..., \mathbf{h}_{v_{|\mathcal{N}(u)}})_{\pi_i} \right)$$

    - ρ_ϕ is a permutation-sensitive function -> usually LSTM
  - Summing over all possible permutations is generally intractable
    - 1. Sample a random subset of possible permutations, and only sum over that random subset
    - 2. Employ a canonical ordering (order the nodes in descending order according to their degree)

Generalized Neighborhood Aggregation

- Neighborhood Attention
  - Graph Attention Network (GAT)

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v$$

$$\alpha_{u,v} = \frac{\exp(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v])}{\sum_{v' \in \mathcal{N}(u)} \exp(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}])}$$

    - α denotes attention weights, $\boldsymbol{a}$ and $\boldsymbol{W}$ are trainable vector or matrix, and $\oplus$ denotes the concatenation operation

Generalized Neighborhood Aggregation

- Variants of attention
  - bilinear attention model

$$\alpha_{u,v} = \frac{\exp(\mathbf{h}_u^\top \mathbf{W} \mathbf{h}_v)}{\sum_{v' \in \mathcal{N}(u)} \exp(\mathbf{h}_u^\top \mathbf{W} \mathbf{h}_{v'})}$$

  - attention layers using MLPs

$$\alpha_{u,v} = \frac{\exp(\mathrm{MLP}(\mathbf{h}_u, \mathbf{h}_v))}{\sum_{v' \in \mathcal{N}(u)} \exp(\mathrm{MLP}(\mathbf{h}_u, \mathbf{h}_{v'})}$$

    - where MLP is restricted to a scalar output

Generalized Neighborhood Aggregation

- Variants of attention
  - Multi-head attention like transformer

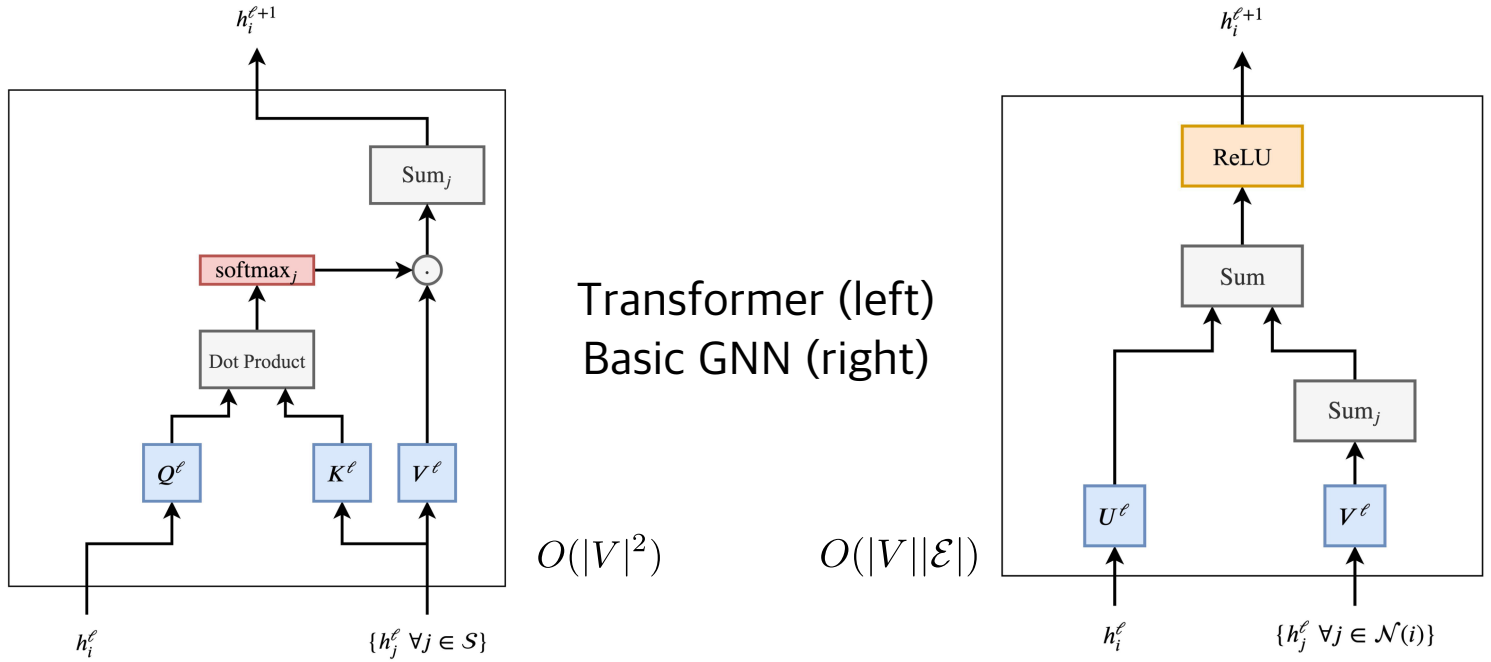
$$\mathbf{m}_{\mathcal{N}(u)} = [\mathbf{a}_1 \oplus \mathbf{a}_2 \oplus ... \oplus \mathbf{a}_K]$$

$$\mathbf{a}_k = \mathbf{W}_k \sum_{v \in \mathcal{N}(u)} \alpha_{u,v,k} \mathbf{h}_v$$

- Evaluation
  - Assumption that some neighbors might be more informative than others -> useful
  - how attention can influence the inductive bias of GNNs -> Chapter 7

Graph attention and transformers

- The basic transformer layer is exactly equivalent to a GNN layer using multi-head attention if we assume that the GNN receives a fully-connected graph as input



Pictures: https://graphdeeplearning.github.io/post/transformers-are-gnns/

Generalized Update Methods

- The AGGREGATE operator has received the most attention after GraphSAGE
- Two key steps of GNN message passing
  - AGGREAGATE
  - UPDATE: defining the power and inductive bias

Over-smoothing and neighborhood influence

- Over-smoothing
  - After several iterations of GNN message passing,
    the representations for all the nodes in the graph can become very similar to one another
  - It makes impossible to build deeper GNN models
- Influence

$$I_K(u, v) = \mathbf{1}^\top \left(\frac{\partial \mathbf{h}_v^{(K)}}{\partial \mathbf{h}_u^{(0)}} \right) \mathbf{1}$$

  - measure of how much the initial embedding of node u influences the final embedding node v

Over-smoothing and neighborhood influence

- Theorem 3 in Xu et al.

  $I_K(u, v) \propto p_{\mathcal{G},K}(u|v)$

  - When we are using a K-layer GCN-style model,
    the influence of node u and node v is proportional the probability of reaching
    node v on a K-step random walk starting from node u
  - As K goes infinity, the influence of every node approaches the stationary distribution
    -> local neighborhood information is lost
  - If $\| \boldsymbol{W}_{self}^{(k)} \| < \| \boldsymbol{W}_{neigh}^{(k)} \|$, this result also applies to the basic GNN.
  - Thus, Deep layers lead to over-smoothing, approaching an almost-uniform distribution

Representation Learning on Graphs with Jumping Knowledge Networks

Generalized Update Methods

- Concatenation and Skip-Connections
  - Using vector concatenations or skip connections alleviates over-smoothing problem
  - Concatenation

    $$\text{UPDATE}_{\text{concat}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = [\text{UPDATE}_{\text{base}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) \oplus \mathbf{h}_u]$$

  - Linear interpolation method

    $$\text{UPDATE}_{\text{interpolate}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \alpha_1 \circ \text{UPDATE}_{\text{base}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) + \alpha_2 \odot \mathbf{h}_2$$

    - where $\alpha_1, \alpha_2 \in [0,1]^d$ are gating vectors with $\alpha_2 = 1 - \alpha_1$
    - the gating parameters can be learned jointly with the model
      - separate single-layer GNN, directly learning for each message passing layer, or MLP

Generalized Update Methods

- Benefits of concatenation and skip-connections
  - help to alleviate the over-smoothing issue
  - improve the numerical stability of optimization
  - facilitate the training of much deeper models like CNNs
  - most useful for node classification tasks
  - excel on tasks that exhibit homophily

Generalized Update Methods

- CNN
  - Concatenation and Skip-Connections
- RNN ?
  - Gating methods:
    update the hidden state of RNN architectures based on observations

Generalized Update Methods

- Gating methods
  - Gated Recurrent Unit (GRU)

    $$\mathbf{h}_u^{(k)} = \mathrm{GRU}(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)})$$

  - LSTM
- Pros
  - very effective at facilitating deep GNN
  - preventing over-smoothing
  - most useful for the prediction task that requires complex reasoning over the global structure

Generalized Update Methods

- Final embedding of the GNN
  - So far, we have been assuming that we are using the output of the final layer of the GNN

$$\boldsymbol{z}_u = \boldsymbol{h}_u^{(K)}, \forall u \in \mathcal{V}$$

  - -> need for residual and gated updates to limit over-smoothing
- Jumping Knowledge Connections

$$\mathbf{z}_u = f_{\mathrm{JK}}(\mathbf{h}_u^{(0)} \oplus \mathbf{h}_u^{(1)} \oplus \ldots \oplus \mathbf{h}_u^{(K)})$$

  - leverage the representations at each layer of message passing
  - f_{JK}: identity, max-pooling, LSTM, ⋯
  - leads to consistent improvements across a wide-variety of tasks

Edge Features and Multi-relational GNNs

- What about multi-relational or heterogenous graph?
- Relational Graph Convolutional Network (RGCN)
  - augment the aggregation function to accommodate multiple relation types

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{\tau \in \mathcal{R}} \sum_{v \in \mathcal{N}_\tau(u)} \frac{\mathbf{W}_\tau \mathbf{h}_v}{f_n(\mathcal{N}(u), \mathcal{N}(v))}$$

    - where f_n is a normalization function
  - analogous to the basic a GNN approach with normalization
  - but just separately aggregate information across different edge types

Edge Features and Multi-relational GNNs

- RGCN
  - drawback: drastic increase in the number of parameters
    -> lead to overfitting and slow learning
- Parameter sharing

$$\mathbf{W}_\tau = \sum_{i=1}^{b} \alpha_{i,\tau} \mathbf{B}_i$$

  - with b basis matrices $\boldsymbol{B}_1, \ldots, \boldsymbol{B}_b$

Edge Features and Multi-relational GNNs

- Parameter sharing
  - the full aggregation function

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{\tau \in \mathcal{R}} \sum_{v \in \mathcal{N}_\tau(u)} \frac{\alpha_\tau \times_1 \mathcal{B} \times_2 \mathbf{h}_v}{f_n(\mathcal{N}(u), \mathcal{N}(v))}$$

    - $\mathcal{B} = (\mathbf{B}_1, ..., \mathbf{B}_b)$ is a tensor formed by stacking the basis matrices
    - $\alpha_\tau = (\alpha_{1,\tau}, ..., \alpha_{b,\tau})$ is a vector containing the basis combination weights for relation τ
    - $\times_i$ denotes a tensor product along mode i.

N-mode product

- The n-mode product between the tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ and the matrix $\boldsymbol{M} \in \mathbb{R}^{J_n \times I_n}$ is defined as

$$\mathcal{B} = \mathcal{A} \times_n \mathbf{M}$$

where the product tensor $\mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N}$

$\mathbf{B}_{(n)} = \mathbf{M}\mathbf{A}_{(n)}$.

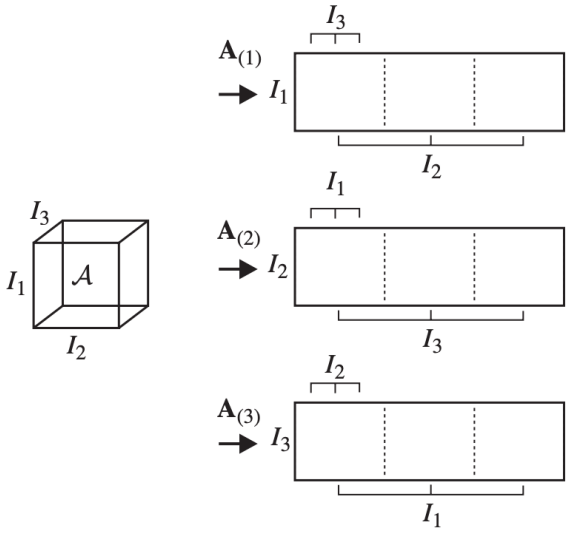


Figure A.1 Example matrix unfolding of a third-order tensor.

AN INTRODUCTION TO TENSOR ALGEBRA

Edge Features and Multi-relational GNNs

- Attention and Feature Concatenation

$$\mathbf{m}_{\mathcal{N}(u)} = \mathrm{AGGREGATE}_{\mathrm{base}}(\{\mathbf{h}_v \oplus \mathbf{e}_{u,\tau,v}, \forall v \in \mathcal{N}(u)\})$$

  - $e_{u,\tau,v}$ denotes an arbitrary vector-valued feature for the edge
  - attention-based approaches as the base aggregation function

Graph Pooling

- What if we want to make predictions at the graph level?
  - Goal: learn an embedding $\boldsymbol{z}_{\mathcal{G}}$ for the entire graph $\mathcal{G}$
  - Task: Graph Pooling
- Set pooling approaches
  - graph pooling can be viewed as a problem of learning over sets

$$\{\mathbf{z}_1, \ldots, \mathbf{z}_{|\mathcal{V}|}\} \overset{maps}{\longrightarrow} \mathbf{z}_{\mathcal{G}}$$

Graph Pooling

- Set pooling approaches
  - Sum (or mean)

$$\mathbf{z}_{\mathcal{G}} = \frac{\sum_{v \in \mathcal{V}} \mathbf{z}_u}{f_n(|\mathcal{V}|)}$$

    - sufficient for applications involving small graphs

Graph Pooling

- Set pooling approaches
  - combination of LSTMs and attention to pool the node embeddings

$$\mathbf{q}_t = \mathrm{LSTM}(\mathbf{o}_{t-1}, \mathbf{q}_{t-1})$$
$$e_{v,t} = f_a(\mathbf{z}_v, \mathbf{q}_t), \forall v \in \mathcal{V}$$
$$a_{v,t} = \frac{\exp(e_{v,i})}{\sum_{u \in \mathcal{V}} \exp(e_{u,t})}, \forall v \in \mathcal{V}$$
$$\mathbf{o}_t = \sum_{v \in \mathcal{V}} a_{v,t} \mathbf{z}_v$$

    - q_t is query vector and $f_a : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is attention function (e.g., dot product)
  - an embedding for the full graph is computed as

$$\mathbf{z}_{\mathcal{G}} = \mathbf{o}_1 \oplus \mathbf{o}_2 \oplus \ldots \oplus \mathbf{o}_T$$

Graph Pooling

- Drawback of Set pooling approaches
  - Set pooling approaches do not exploit the structure of the graph (only using node feature)
- Graph coarsening (or clustering) approaches
  - exploit the graph topology at the pooling stage

Graph Pooling

- Graph coarsening approaches
  - we assume that we have some clustering function

$$\mathbf{f}_c \rightarrow \mathcal{G} \times \mathbb{R}^{|\mathcal{V}| \times d} \rightarrow \mathbb{R}^{+|\mathcal{V}| \times c}$$

    - which maps all the nodes to an assignment over c clusters
    - this function outputs an assignment matrix $\mathbf{S} = f_c(\mathcal{G}, \mathbf{Z})$,
      where $\mathbf{S}[u, i] \in \mathbb{R}^+$ denotes the strength of the association between node u and cluster i
    - f_c : spectral clustering approach (Chapter 1), another GNN to predict cluster assignments, ⋯

Graph Pooling

- Graph coarsening approaches
  - the assignment matrix S is used to coarsen the graph

$$\mathbf{A}^{\text{new}} = \mathbf{S}^{\top}\mathbf{A}\mathbf{S} \in \mathbb{R}^{+c \times c}$$

$$\mathbf{X}^{\text{new}} = \mathbf{S}^{\top}\mathbf{X} \in \mathbb{R}^{c \times d}$$

  - the new adjacency matrix represents the strength of association between the clusters in the graph
  - the new feature matrix represents the aggregated embeddings for all the nodes assigned to each cluster
  - we can repeat the entire coarsening process a several times
  - Final representation = set pooling over a sufficiently coarsened graph

Graph Pooling

- Graph coarsening approaches
  - can lead to strong performance
  - can also be unstable and difficult to train
    - To be end-to-end, the clustering algorithms must be differentiable
    - But it rules out most off-the-shelf clustering algorithms such as spectral clustering
  - Another approach: selecting a set of nodes to remove rather than pooling all nodes

Generalized Message Passing

- Each iteration of message passing

$$\mathbf{h}_{(u,v)}^{(k)} = \text{UPDATE}_{\text{edge}}(\mathbf{h}_{(u,v)}^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}, \mathbf{h}_{\mathcal{G}}^{(k-1)})$$

$$\mathbf{m}_{\mathcal{N}(u)} = \text{AGGREGATE}_{\text{node}}(\{\mathbf{h}_{(u,v)}^{(k)} \forall v \in \mathcal{N}(u)\})$$

$$\mathbf{h}_u^{(k)} = \text{UPDATE}_{\text{node}}(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}, \mathbf{h}_{\mathcal{G}}^{(k-1)})$$

$$\mathbf{h}_{\mathcal{G}}^{(k)} = \text{UPDATE}_{\text{graph}}(\mathbf{h}_{\mathcal{G}}^{(k-1)}, \{\mathbf{h}_u^{(k)}, \forall u \in \mathcal{V}\}, \{\mathbf{h}_{(u,v)}^{(k)} \forall (u,v) \in \mathcal{E}\})$$

1. The edge embeddings are updated based on the embeddings of their incident nodes
2. The node embeddings are updated by aggregating the edge embeddings for all their incident edges
3. The graph embedding is updated by aggregating over all the node and edge embeddings

Thanks!

- Any question so far?