Machine Learning with Graphs (MLG)

Graph Neural Networks (GNN)

[GNN三劍客] GCN, GraphSage, GAT

Cheng-Te Li (李政德)

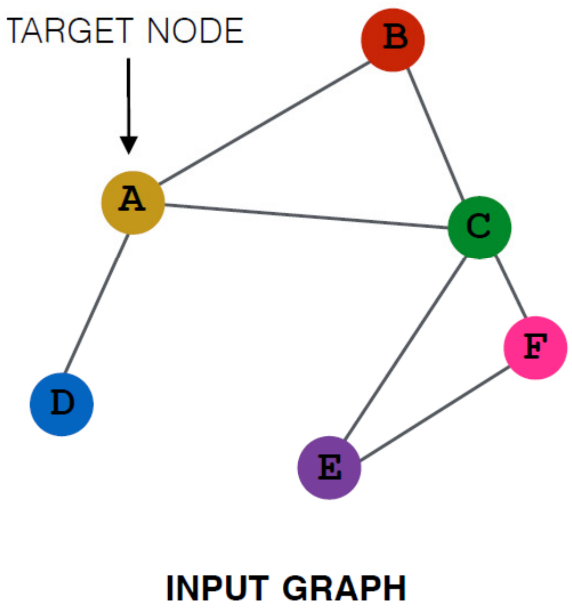
Institute of Data Science

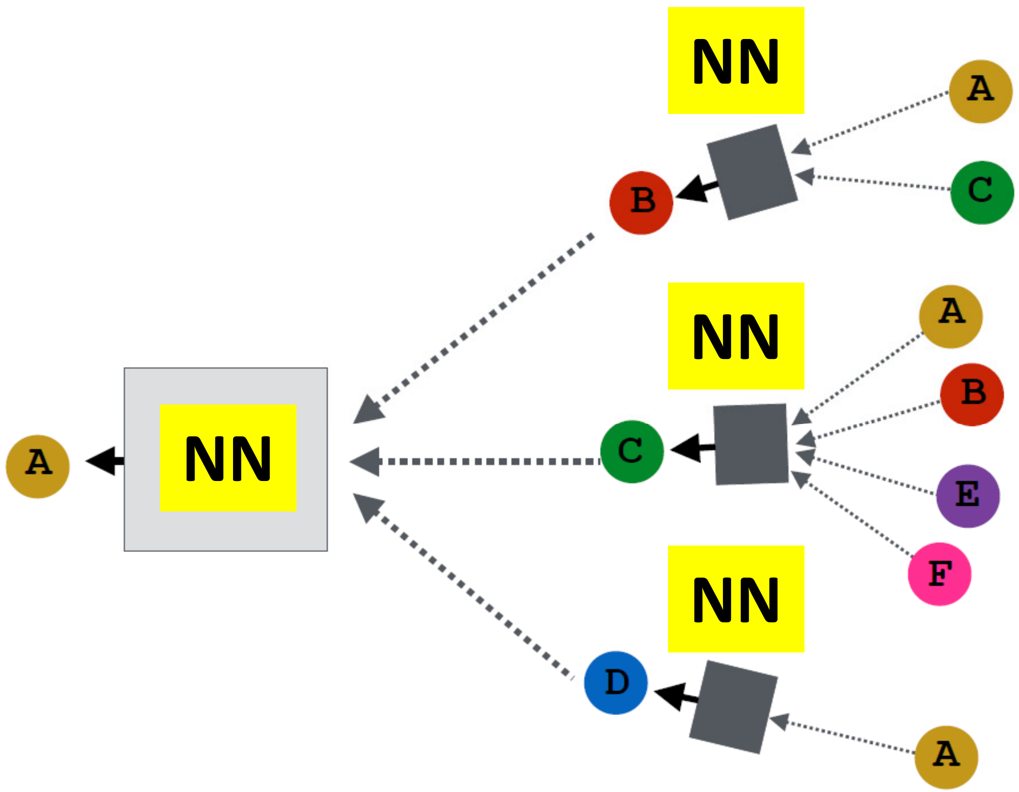
National Cheng Kung University

chengte@mail.ncku.edu.tw

Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate messages
- What else can we implement the **NN**?





Graph Convolutional Networks (**GCN**)

- A slight variation on the neighborhood aggregation

Basic Neighborhood Aggregation

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

VS.

GCN Neighborhood Aggregation

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

same matrix for self and neighbor embeddings

Per-neighbor normalization

Graph Convolutional Networks (**GCN**)

- Empirically, they found this configuration to give better results
  - More parameter sharing
  - Down-weights high degree neighbors

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

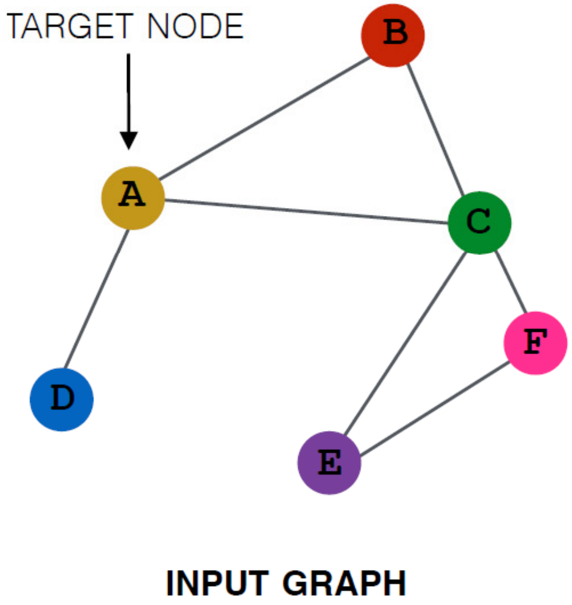
use the same transformation matrix for self and neighbor embeddings

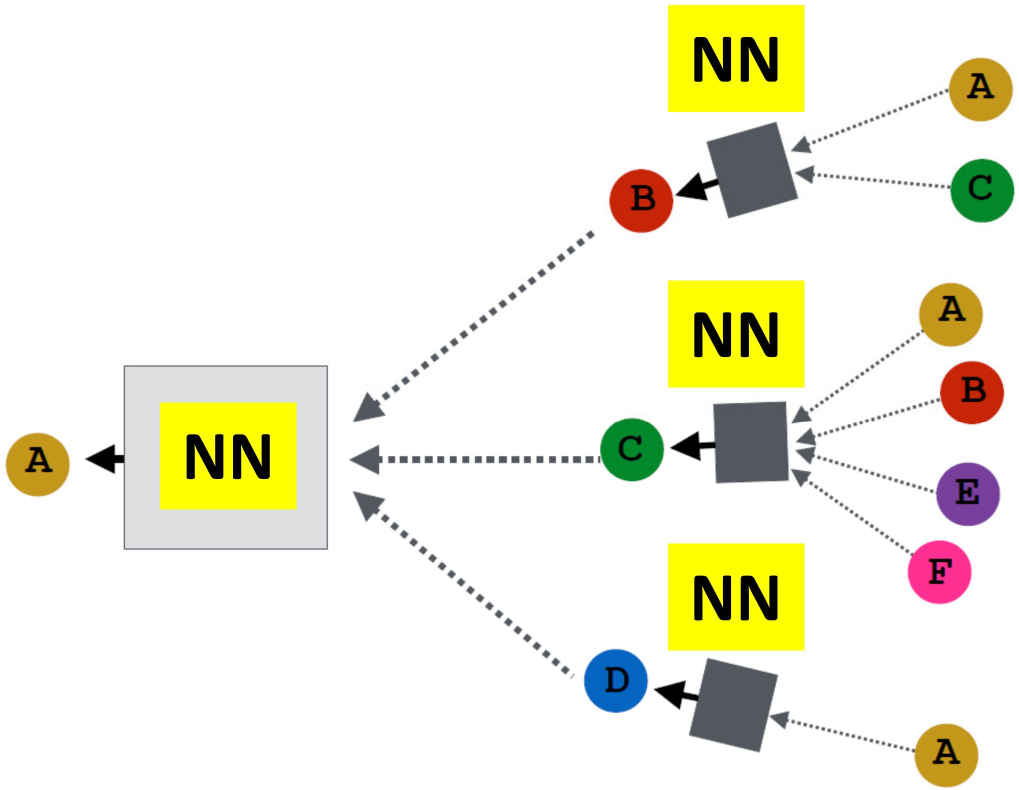
instead of simple average, normalization varies across neighbors

Neighborhood Aggregation

So far we have aggregated the neighbor messages by taking their (weighted) average!

Can we do better?



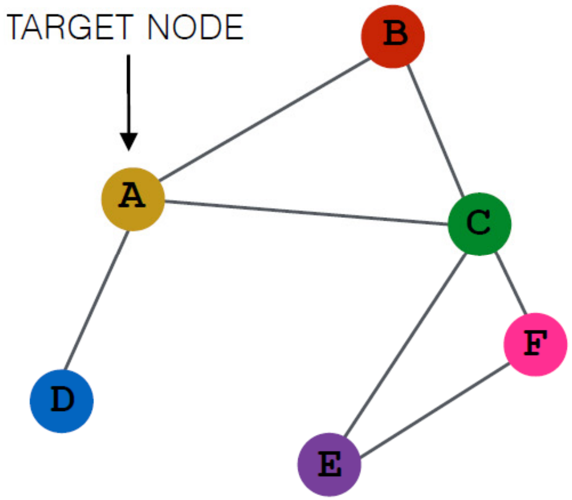


GraphSage Idea

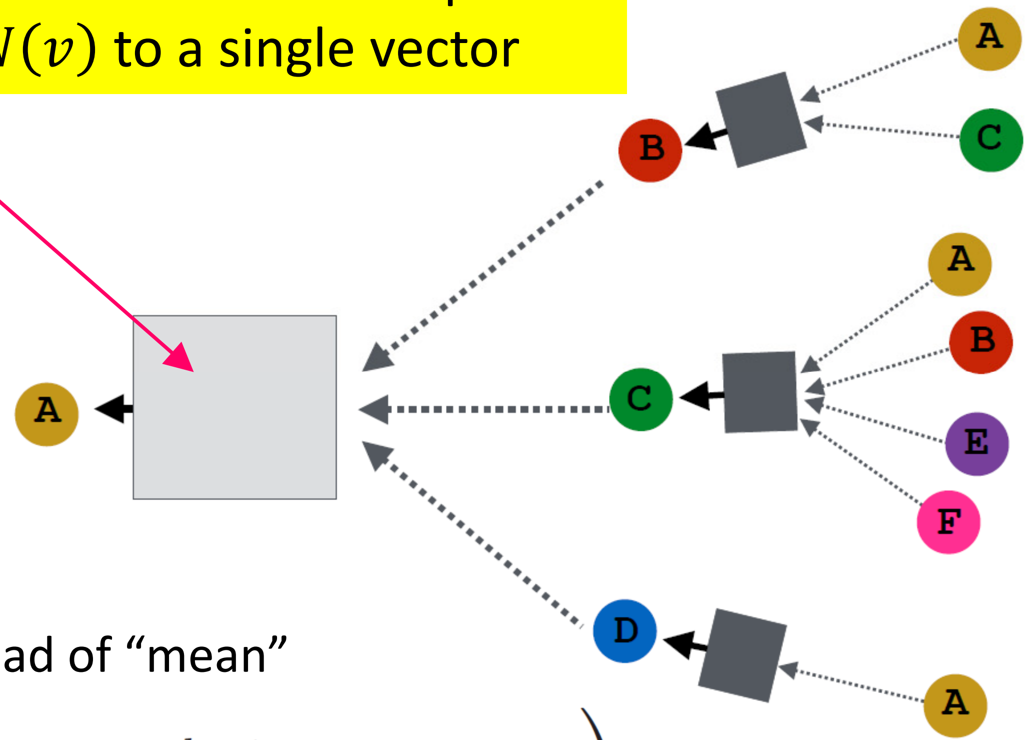
Concat.

$$h_v^{(l+1)} = \sigma\left(\left[W_l \cdot \boxed{\mathrm{AGG}\left(\left\{h_u^{(l)}, \forall u \in N(v)\right\}\right)}, B_l h_v^{(l)}\right]\right)$$

Any differentiable function that maps set of vectors in $N(v)$ to a single vector



INPUT GRAPH



Instead of "mean"

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

vs. Simple neighborhood aggregation

GraphSage Idea

Concat.

$$h_v^{(l+1)} = \sigma\left(\left[W_l \cdot \boxed{\mathrm{AGG}\left(\left\{h_u^{(l)}, \forall u \in N(v)\right\}\right)}, B_l h_v^{(l)}\right]\right)$$

Optional: Apply L2 normalization to $h_1^{(l+1)}$ embedding at every layer

- **L2 Normalization**: $h_v^k \leftarrow \dfrac{h_v^k}{\|h_v^k\|_2}, v \in V$
  - $\|x\|_2 = \sqrt{\sum_i x_i^2}$ (ℓ_2-norm)
  - Without ℓ_2 normalization, the embedding vectors have different scales 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

AGG Variants in GraphSage

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

$$\mathrm{AGG} = \sum_{u \in N(v)} \frac{\mathrm{h}_u^{(l)}}{|N(v)|}$$

- **Pool:** Transform neighbor vectors and apply symmetric vector function

Element-wise mean/max

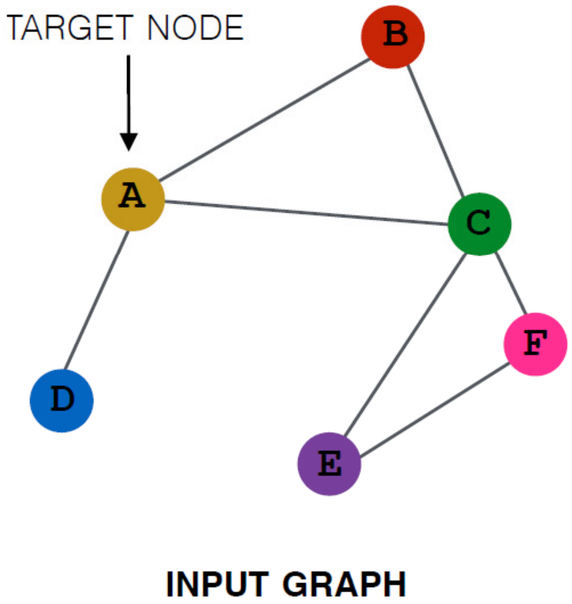
$$\mathrm{AGG} = \boxed{\gamma}(\{\mathrm{MLP}(\mathrm{h}_u^{(l)}), \forall u \in N(v)\})$$

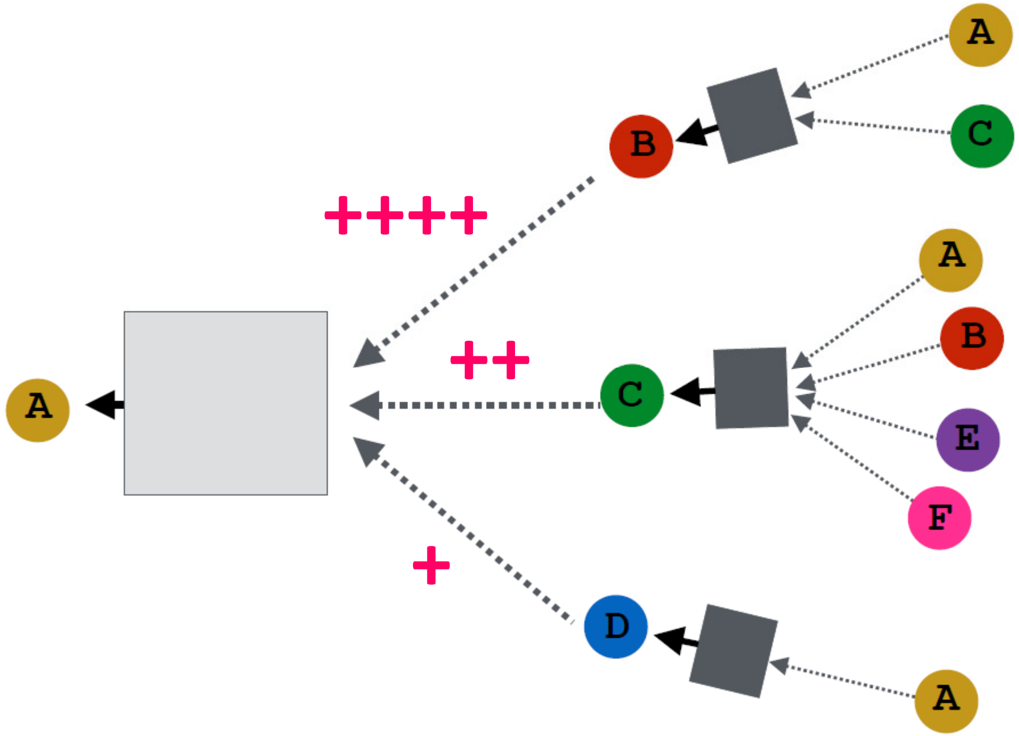
- **LSTM:** Apply LSTM to reshuffled of neighbors

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

Neighborhood Attention

- What if some neighbors are more important than others?





Graph Attention Networks (GAT)

- Augment basic graph neural network model with attention mechanism

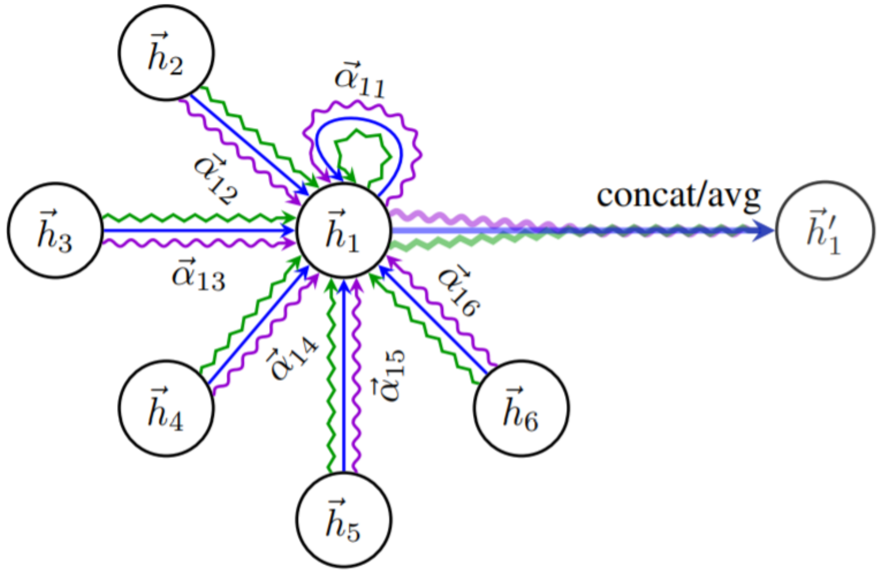
$$h_v^{(l+1)} = \boxed{\sigma}\left(\boxed{\sum_{u \in N(v) \cup \{v\}}} \boxed{\alpha_{v,u}} W^{(l+1)} h_u^{(l)}\right)$$

Non-linearity

Sum over all neighbors (and the node itself)

Learned attention weights

Multi-head attention (#heads = 3 here)

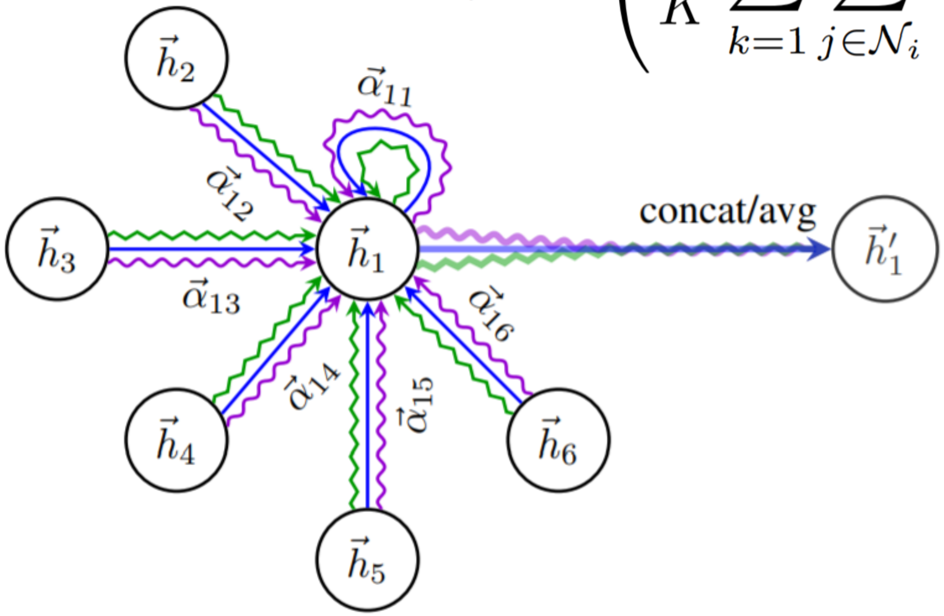


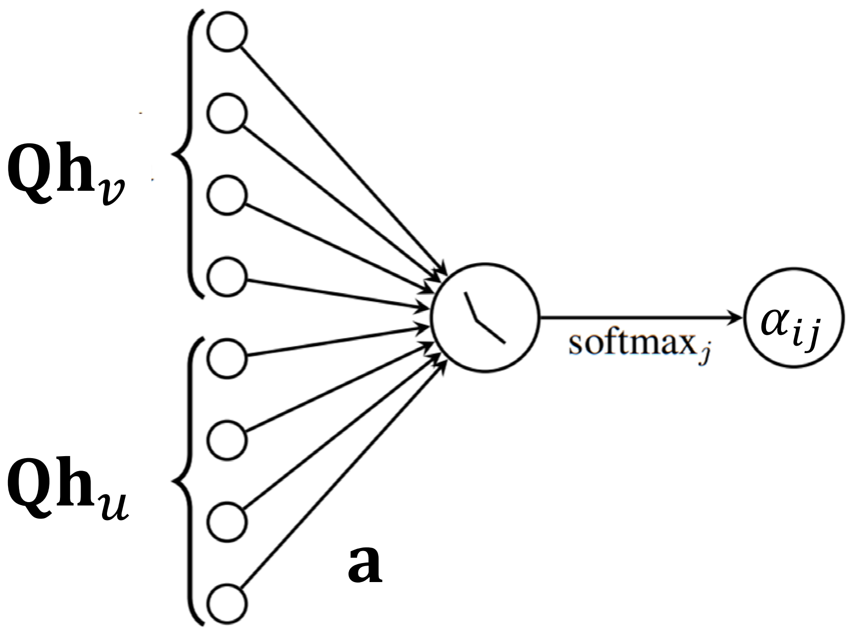
Attention Weights in GAT

- Various attention models are possible
- The original GAT paper uses:

$$\alpha_{v,u} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{Q}\mathbf{h}_v, \mathbf{Q}\mathbf{h}_u]\right)\right)}{\sum_{u' \in N(v)\cup\{v\}} \exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{Q}\mathbf{h}_v, \mathbf{Q}\mathbf{h}_{u'}]\right)\right)}$$

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$





GNN Summary

- Graph Neural Networks borrow ideas from classical neural networks and generalize them to graphs
- Locality
  - Learn local patterns of the graph
- Weight Sharing
  - Learn universally applicable patterns of the graph

A **Graph Neural Network** updates node representations by repeatedly **transforming** and **aggregating neighboring node representations**

Message Passing Scheme

- Each neighbor sends a **message**: $\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$

$$\mathbf{m}_{w,v}^{(\ell)} = \mathrm{MESSAGE}(\mathbf{h}_v^{(\ell-1)}, \mathbf{h}_w^{(\ell-1)})$$

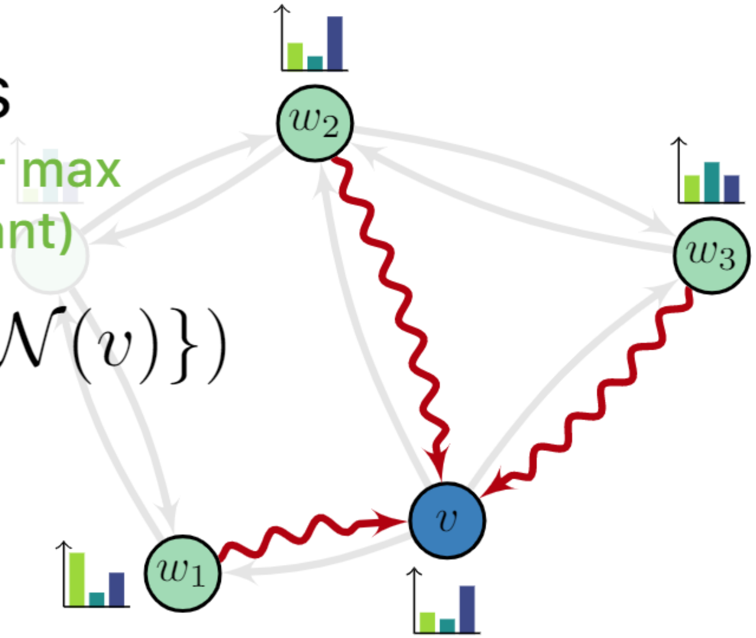
- Messages are **aggregated** across all neighbors:

typically sum, mean or max (permutation-invariant)

$$\mathbf{a}_v^{(\ell)} = \mathrm{AGGREGATE}(\{\mathbf{m}_{w,v}^{(\ell)} : w \in \mathcal{N}(v)\})$$

- Neighbor information is used to **update** node represenation:

$$\mathbf{h}_v^{(\ell)} = \mathrm{UPDATE}(\mathbf{h}_v^{(\ell-1)}, \mathbf{a}_v^{(l)})$$



Message passing functions are trainable and differentiable

Variants on Massage Passing Scheme

- Plain messages $h_v^{(l)}$, $h_w^{(l)}$
- Before **AGG**regation and **UPDATE** ...

It can model **anisotropic transformations:**

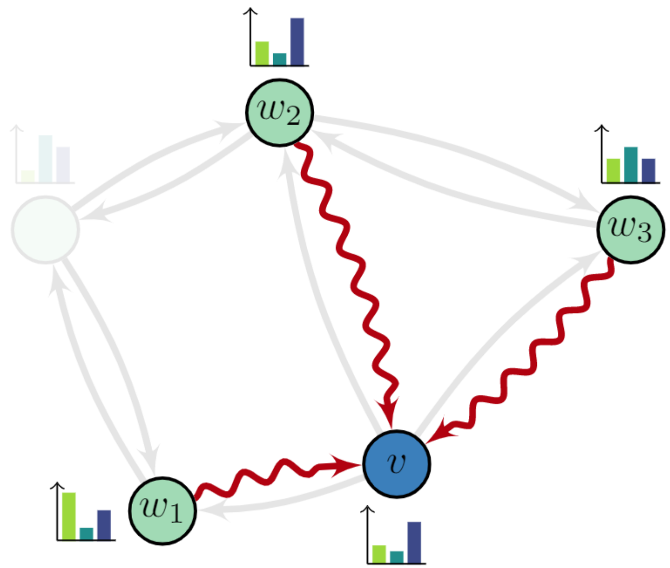
$$\mathrm{MESSAGE}(\mathbf{h}_v^{(\ell)}, \mathbf{h}_w^{(\ell)}) = \mathrm{MLP}(\mathbf{h}_v^{(\ell)}, \mathbf{h}_w^{(\ell)} - \mathbf{h}_v^{(\ell)})$$

It can model different **edge relation types:**

$$\mathrm{MESSAGE}(\mathbf{h}_v^{(\ell)}, \mathbf{h}_w^{(\ell)}) = \mathbf{W}_{R(v,w)}\mathbf{h}_w^{(\ell)}$$

It can incorporate **edge features:**

$$\mathrm{MESSAGE}(\mathbf{h}_w^{(\ell)}, \mathbf{e}_{w,v}) = \gamma(\mathbf{e}_{w,v})\mathbf{h}_w^{(\ell)}$$



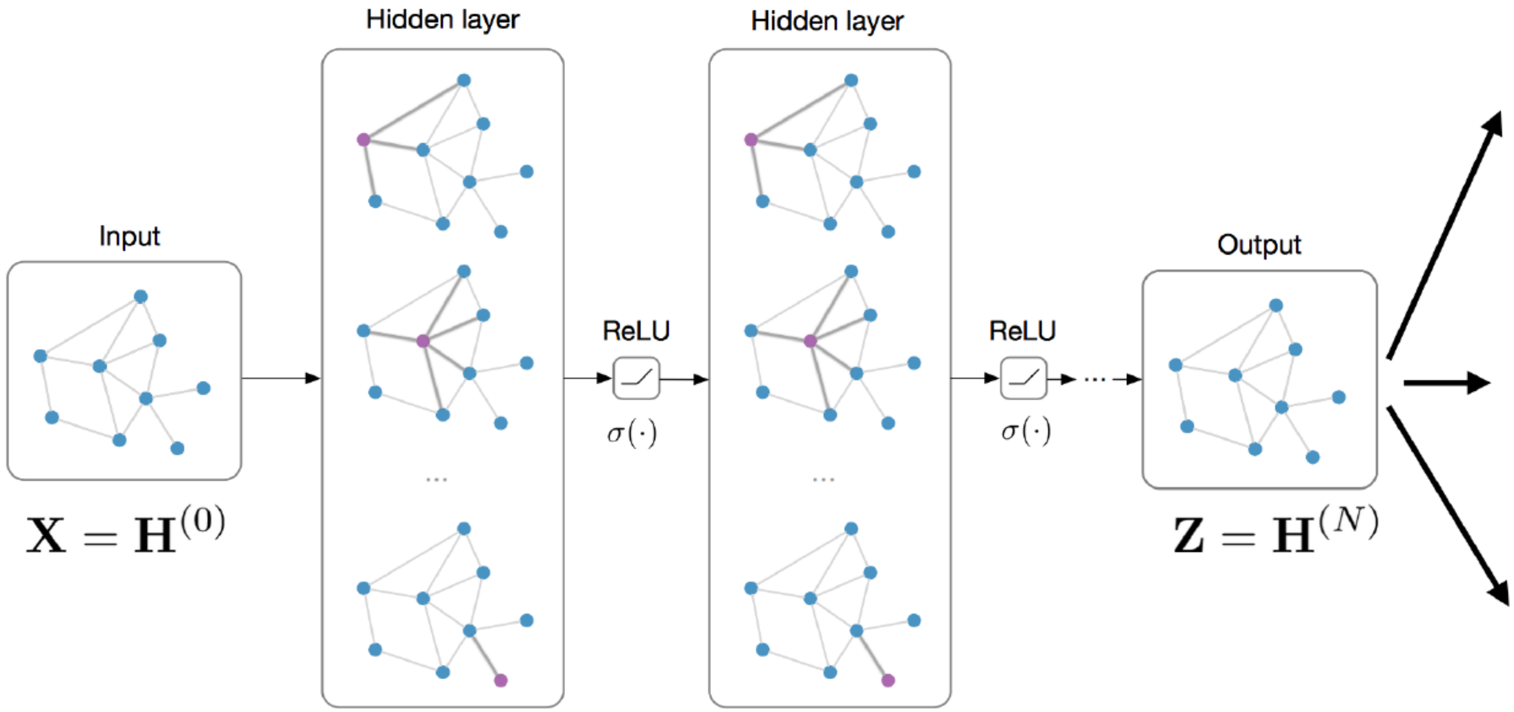
Wang et al.: Dynamic Graph CNN for Learning on Point Clouds? ('19)
Schlichtkrull et al.: Modeling Relational Data with Graph Convolutional Networks ('17)
Fey et al.: SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels (CVPR '18)

Advantages of GNNs

- #of parameters is independent of graph size
  - The parameters in the GNN layers depend only on the dimension of the feature inputs
- Inherently "inductive"
  - After training GNNs can be used to infer embeddings on unseen graphs
- Naturally incorporate node features
  - GNNs learn by aggregating node features over local neighborhoods

Tasks with GNNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Node classification:

$$\mathrm{softmax}(\mathbf{z_n})$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$\mathrm{softmax}(\textstyle\sum_n \mathbf{z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

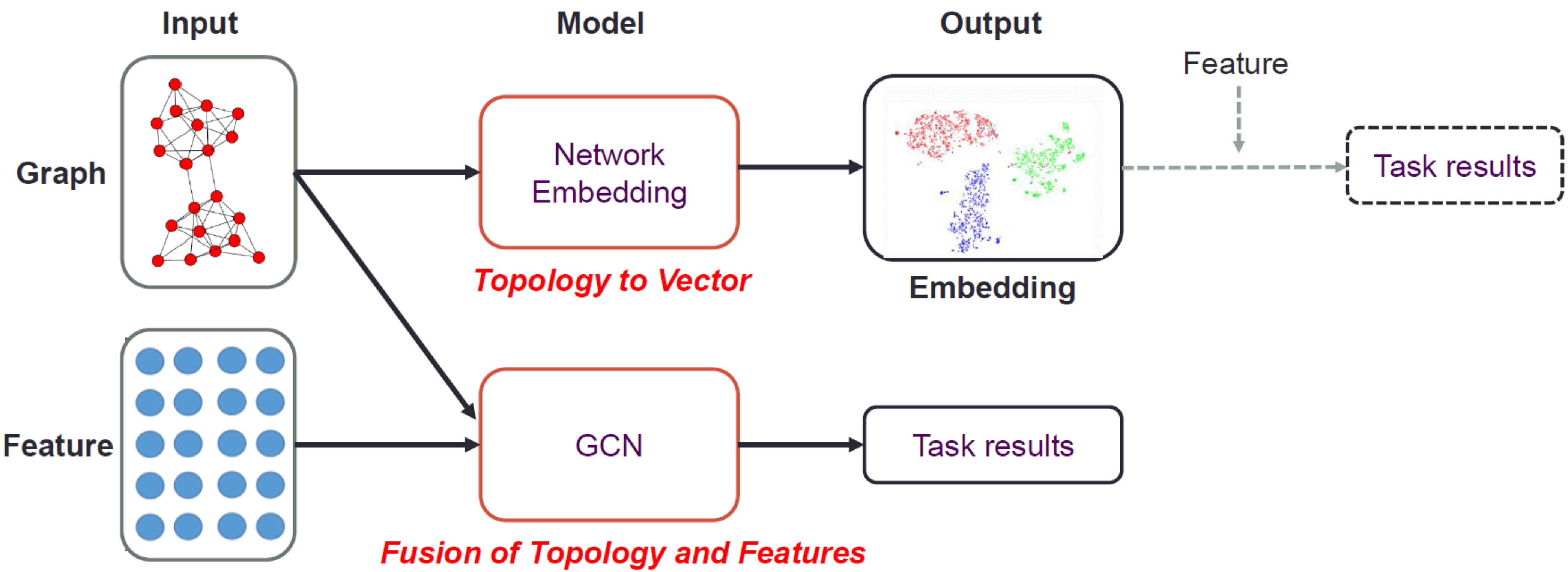
Link prediction:

$$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$$

Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

GRL vs. GNN



Unsupervised vs. (Semi-)Supervised

[GNN三劍客] References

- T. Kipf & M. Welling "**Semi-Supervised Classification with Graph Convolutional Networks**" ICLR 2017   7989 cites
- P. Velickovic et al. "**Graph Attention Networks**" ICLR 2018   3625 cites
- W. L. Hamilton et al. "**Inductive Representation Learning on Large Graphs**" NIPS 2017   3366 cites

PyTorch geometric

A Python library built upon PyTorch to enable deep learning on graphs

- Simplify implementing and working with Graph Neural Networks
- Omit the need of re-writing common logic (operators, datasets, batching, ...)
- Bundle fast implementations from published papers

GNNs	Cheby GCN SAGE PointNet MoNet MPNN GAT SplineCNN AGNN EdgeCNN S-GCN R-GCN PointCNN ARMA APPNP GIN GIN-E CG GatedGCN NMF TAG Signed-GCN DNA PPFNet FeaST Hyper-GCN GravNet
Pooling	Set2Set SortPool DiffPool MinCUT Graclus VoxelGrid TopK SAG EdgePool ASAP
Models	(V)GAE ARG(V)A DGI Node2Vec GraphUNet GeniePath SchNet DimeNet MetaPath2Vec ReNet
Utilities	GNNExplainer SIGN GDC ClusterGCN DropEdge GraphSAINT NeighborSampling GraphSizeNorm JK