Deep Learning

Graph Neural Network

U Kang
Seoul National University

In This Lecture

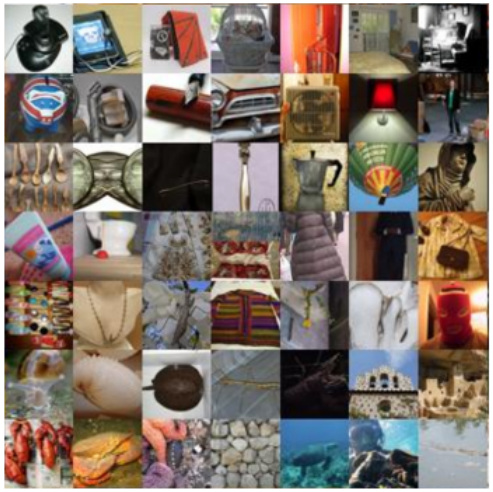
- Motivation of graph deep learning
- Graph neural networks
- Applications of GNN

Outline

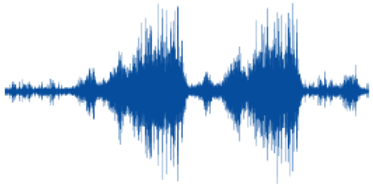
- **Motivation**
- Graph Neural Network
- Applications

Deep Learning

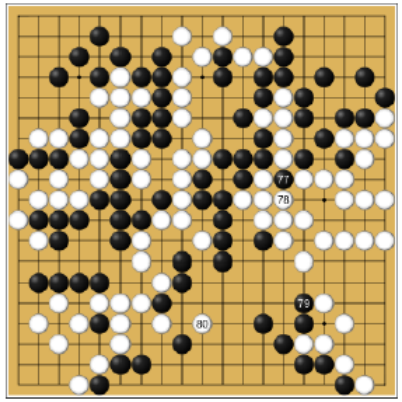
IMAGENET



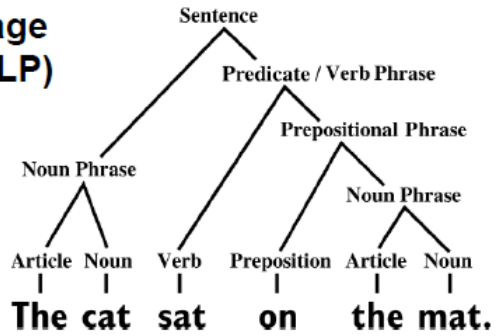
Speech data



Grid games



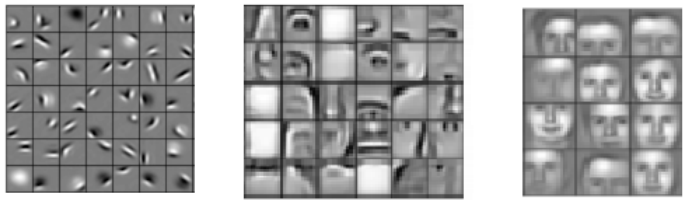
Natural language processing (NLP)



...

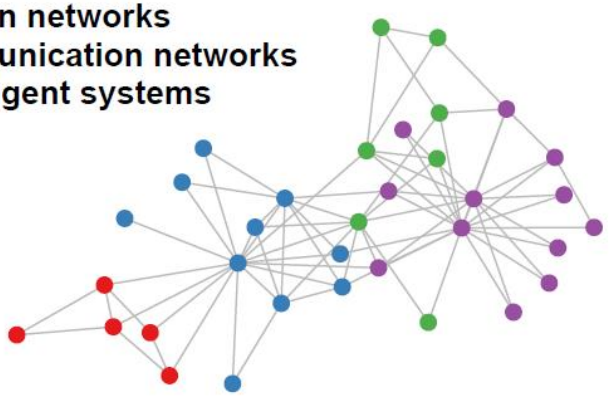
Deep neural nets that exploit:

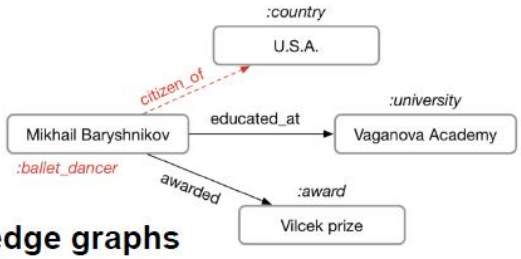
- translation equivariance (weight sharing)
- hierarchical compositionality



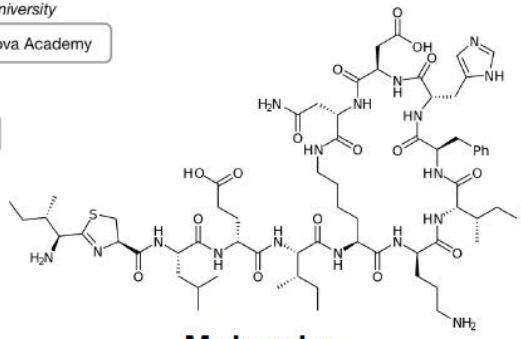
Graph Structured Data

Social networks
Citation networks
Communication networks
Multi-agent systems

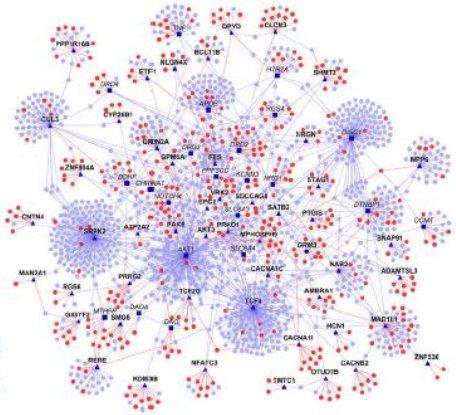




Knowledge graphs



Molecules



Protein interaction networks

Road maps

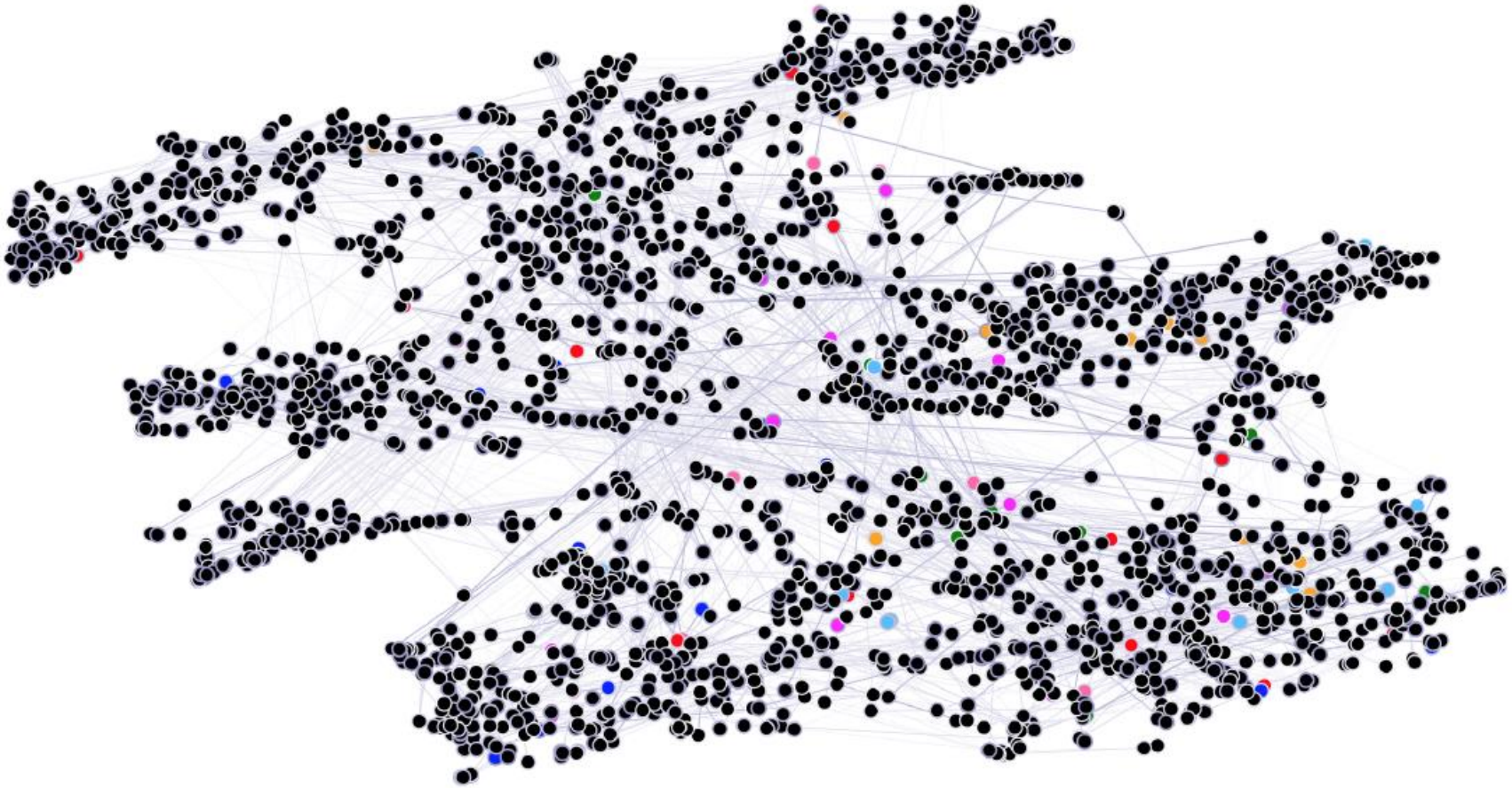
- Standard deep models (CNN, RNN) don't work

Formulation

- Node classification problem
  - Input: a matrix of node features, $X \in R^{N \times C}$, with C features in each of the N nodes, and an adjacency matrix $A \in R^{N \times N}$
  - Output: a matrix of node class probabilities, $Y \in R^{N \times F}$, such that $Y_{ij} = P(node\ i\ \in class\ j)$
- We assume that the edges are unweighted and undirected
  - $A_{ij} = A_{ji} = \begin{cases} 1 & i \leftrightarrow j \\ 0 & otherwise \end{cases}$

Learning: Transductive vs. Inductive

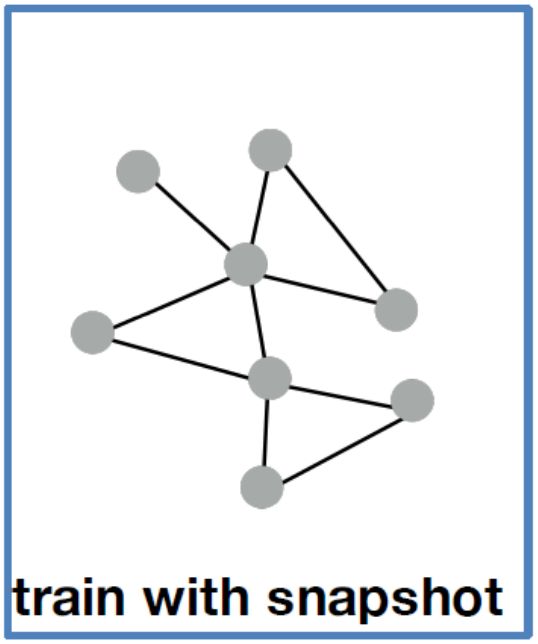
- Transductive: training algorithm sees all the features, including test nodes

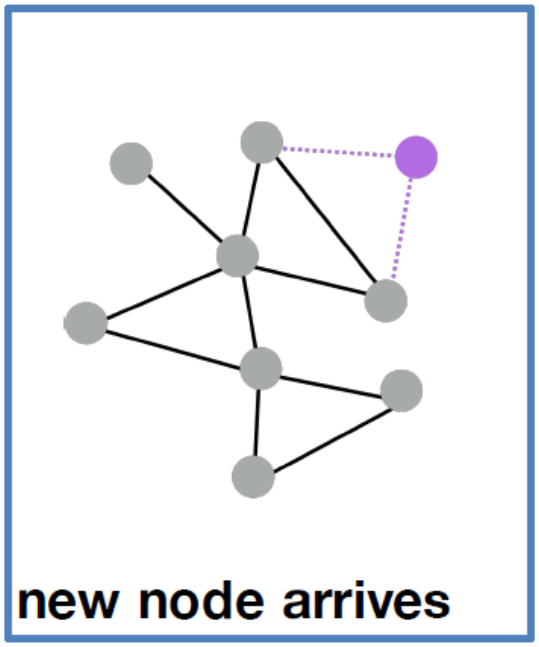


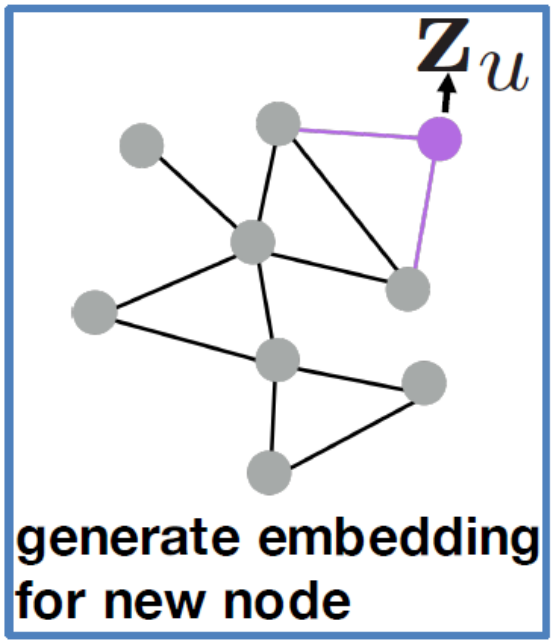
Learning: Transductive vs. Inductive

- Inductive learning
  - The algorithm does not have access to all nodes upfront!
  - This implies either
    - Test nodes are (incrementally) inserted into training graphs
    - Test graphs are disjoint and completely unseen
  - A much harder learning problem (requires generalizing across arbitrary graph structures), and many transductive methods will be inappropriate for inductive problems

Inductive Learning







Outline

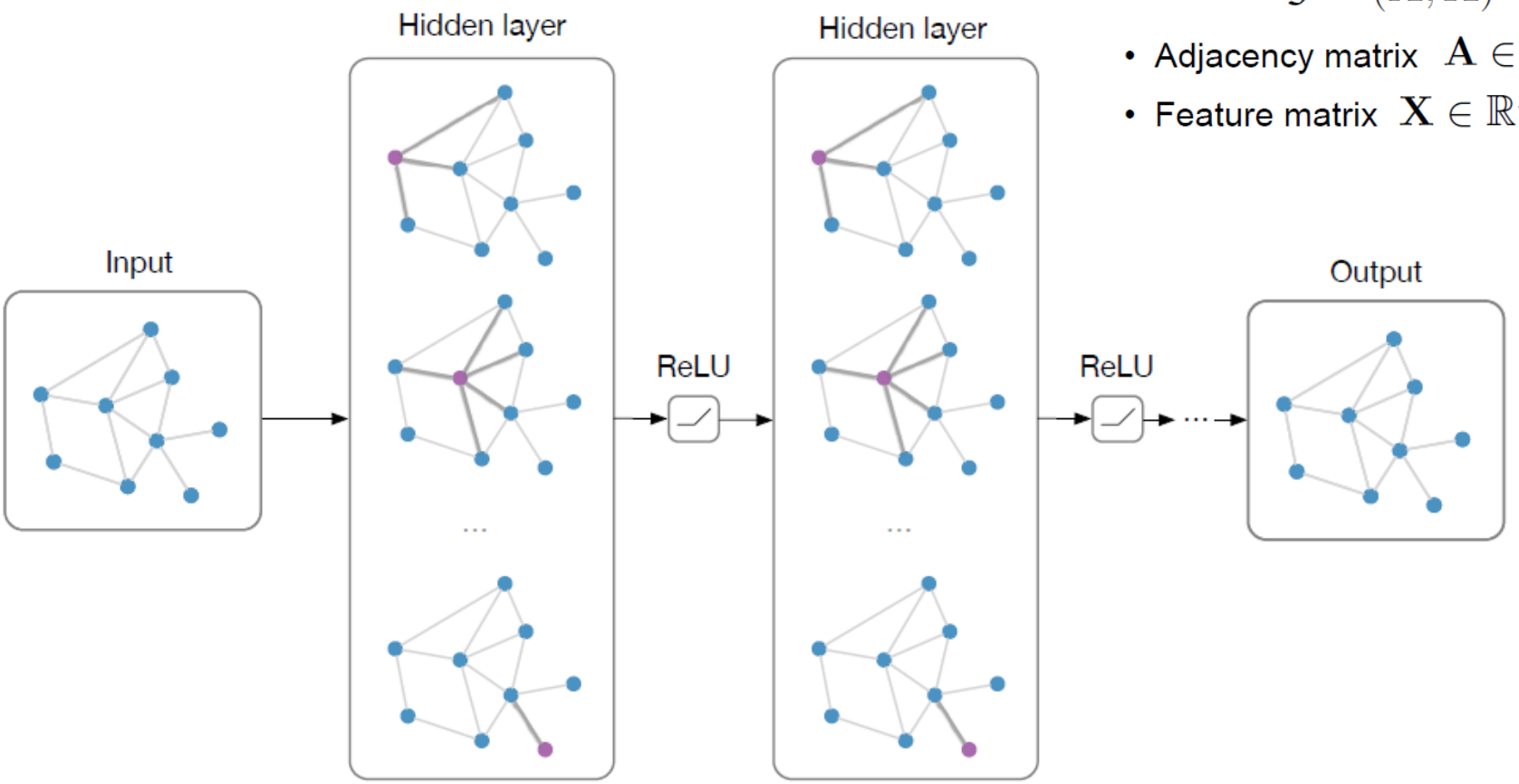
- ☑ Motivation
- ☐ **Graph Neural Network**
  - **GCN**
  - GraphSAGE
  - GAT
  - Gated GNN
- ☐ Applications

Graph Neural Networks (GNNs)

- Main idea: pass messages between pairs of nodes and agglomerate

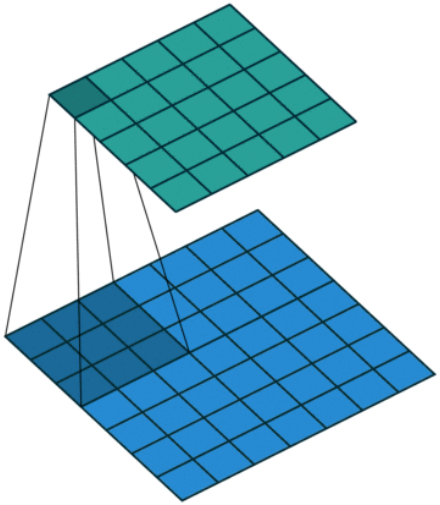
Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$

- Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$

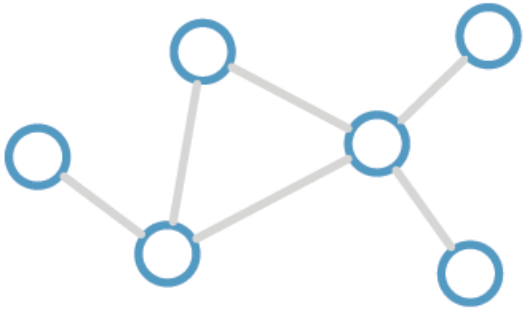


[Kipf & Welling (ICLR 2017)]

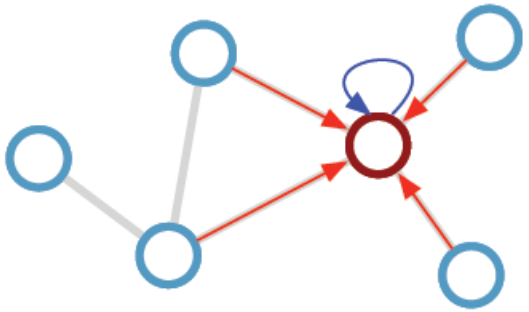
Graph Convolutional Networks (GCNs)



Consider this undirected graph:



Calculate update for node in red:



GCN

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

Example: Citation Network

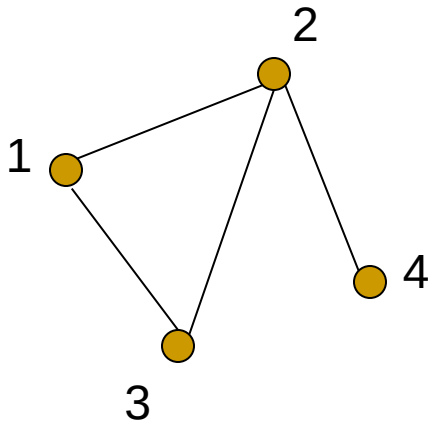
Input: Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

Target: Paper category (e.g. stat.ML, cs.LG, …)

Model: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$

- $\tilde{A} = A + I_N, \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
- $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} \in R^{N\times N}$
- $X \in R^{N\times C}$
- $W^{(0)} \in R^{C\times H}$
- $W^{(1)} \in R^{H\times F}$

GCN Example: Citation Network



1	1	1	
1	1	1	1
1	1	1	
	1		1

$\tilde{A}$

3			
	4		
		3	
			2

$\tilde{D}$

1/3	1/3	1/3	
1/4	1/4	1/4	1/4
1/3	1/3	1/3	
	1/2		1/2

$\tilde{D}^{-1}\tilde{A}$

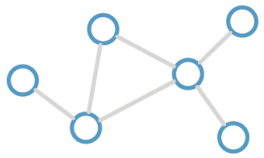
$\frac{1}{\sqrt{9}}$	$\frac{1}{\sqrt{12}}$	$\frac{1}{\sqrt{9}}$	
$\frac{1}{\sqrt{12}}$	$\frac{1}{\sqrt{16}}$	$\frac{1}{\sqrt{12}}$	$\frac{1}{\sqrt{8}}$
$\frac{1}{\sqrt{9}}$	$\frac{1}{\sqrt{12}}$	$\frac{1}{\sqrt{9}}$	
	$\frac{1}{\sqrt{8}}$		$\frac{1}{\sqrt{4}}$

$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

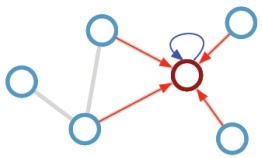
[Kipf & Welling (ICLR 2017)]

Graph Convolutional Networks (GCNs)

Consider this undirected graph:



Calculate update for node in red:

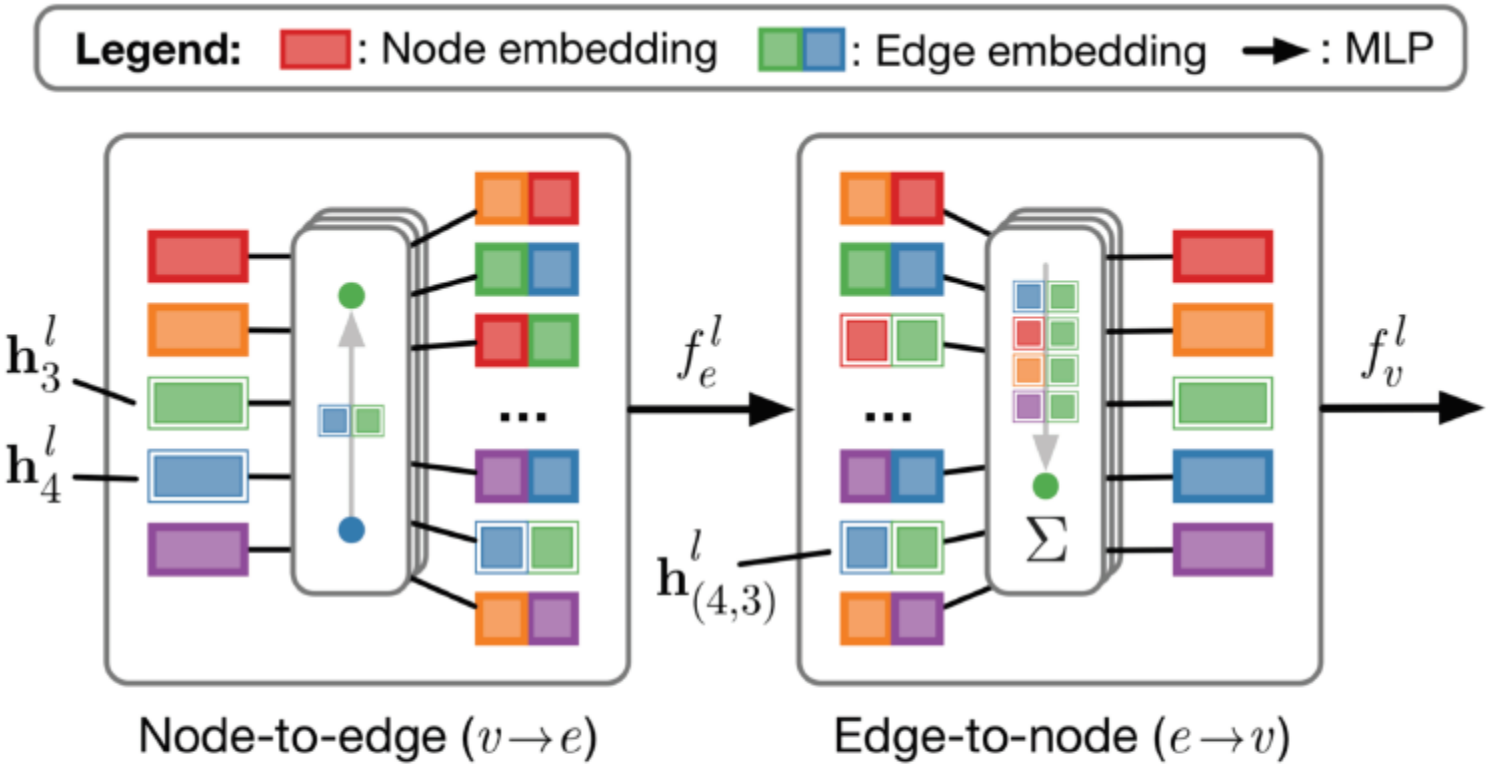


Update rule: $\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$

- Pros
  - Weight sharing over all locations
  - Invariance to permutations
  - Linear complexity O(E)
  - Applicable both in transductive and inductive settings (but, limited in inductive setting)
- Cons
  - Requires residual connections for depth (oversmoothing problem)
  - Only indirect support for edge features

[Kipf et al. (ICML 2018)]

GNNs with Edge Embeddings



Formally:

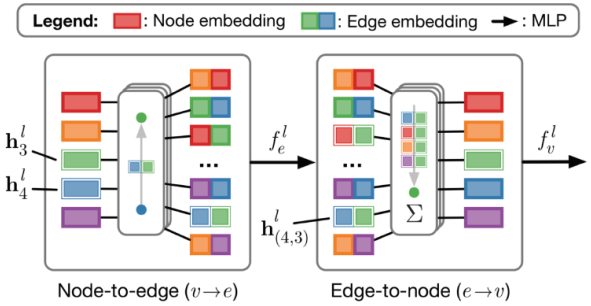
$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

[Kipf et al. (ICML 2018)]

GNNs with Edge Embeddings

- Pros
  - Supports edge features
  - More expressive than GCN
  - As general as it gets (?)



Formally:

$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

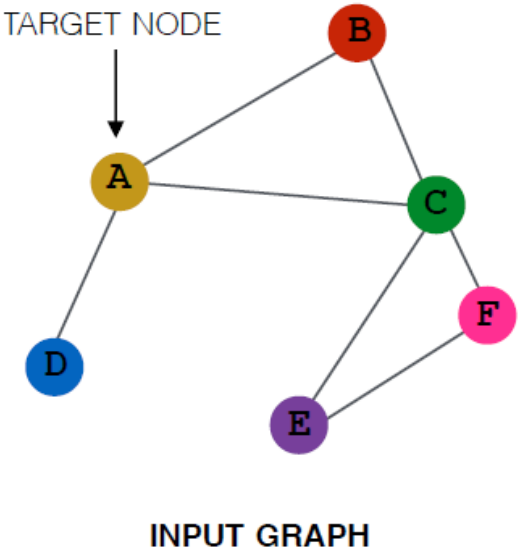
- Cons
  - Need to store intermediate edge-based activations

Outline

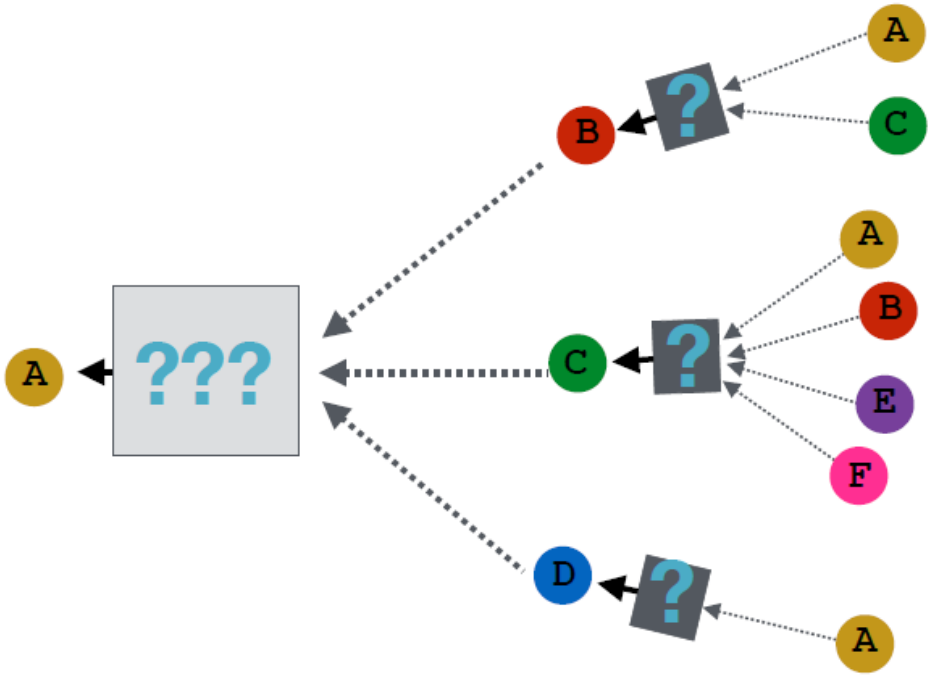
- ☑ Motivation
- ☐ **Graph Neural Network**
  - GCN
  - **GraphSAGE**
  - GAT
  - Gated GNN
- ☐ Applications

Motivation of GraphSAGE

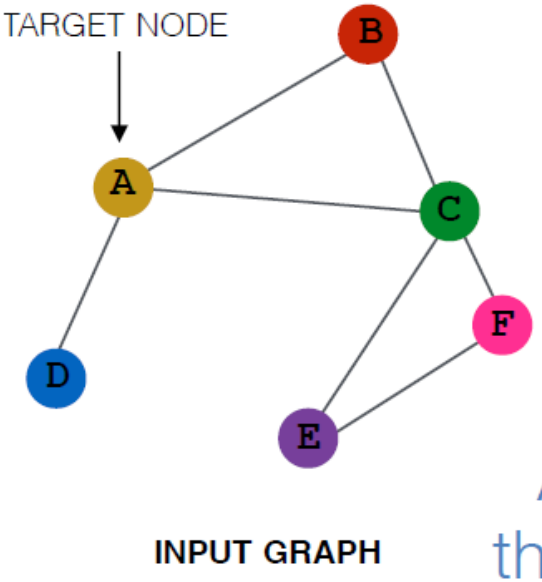
- We have integrated the neighbor messages by taking their weighted average; can we do better?



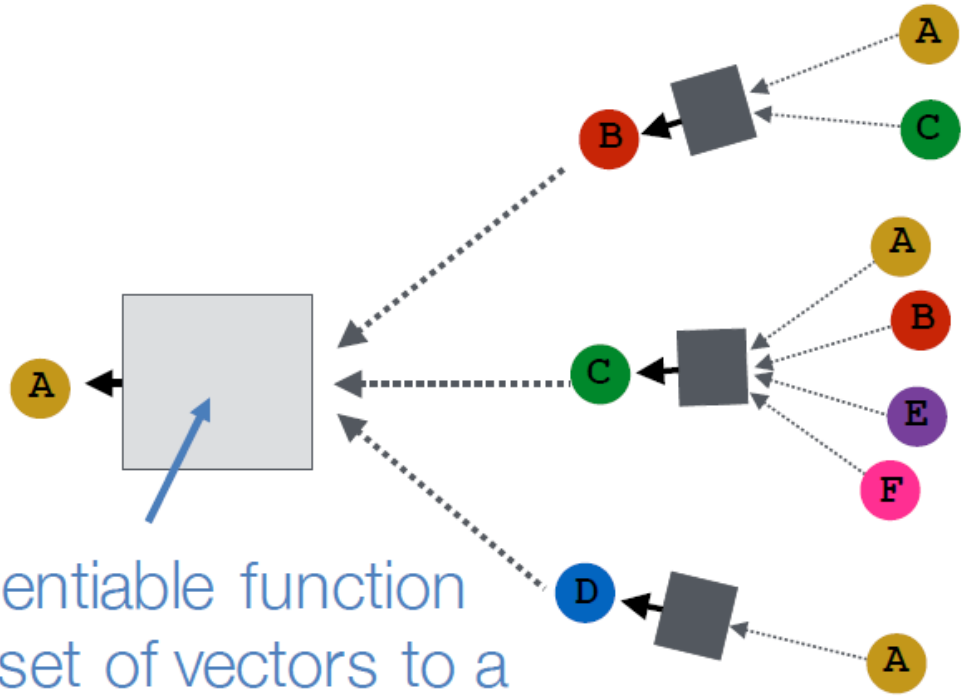
INPUT GRAPH



Motivation of GraphSAGE



INPUT GRAPH



Any differentiable function that maps set of vectors to a single vector.

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{A}_k \cdot \text{AGG}(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}), \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

GraphSAGE

- Before: Simple neighborhood aggregation

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

- GraphSAGE
  - Concatenate self embedding and neighbor embedding

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{W}_k \cdot \mathrm{AGG}\left(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right), \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

generalized aggregation

GraphSAGE Variants

- Mean

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

- Pool: transform neighbor vectors and apply symmetric vector function

element-wise mean/max

$$\text{AGG} = \gamma\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$$

- LSTM: apply LSTM to random permutation of neighbors

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

Experiment

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

Outline

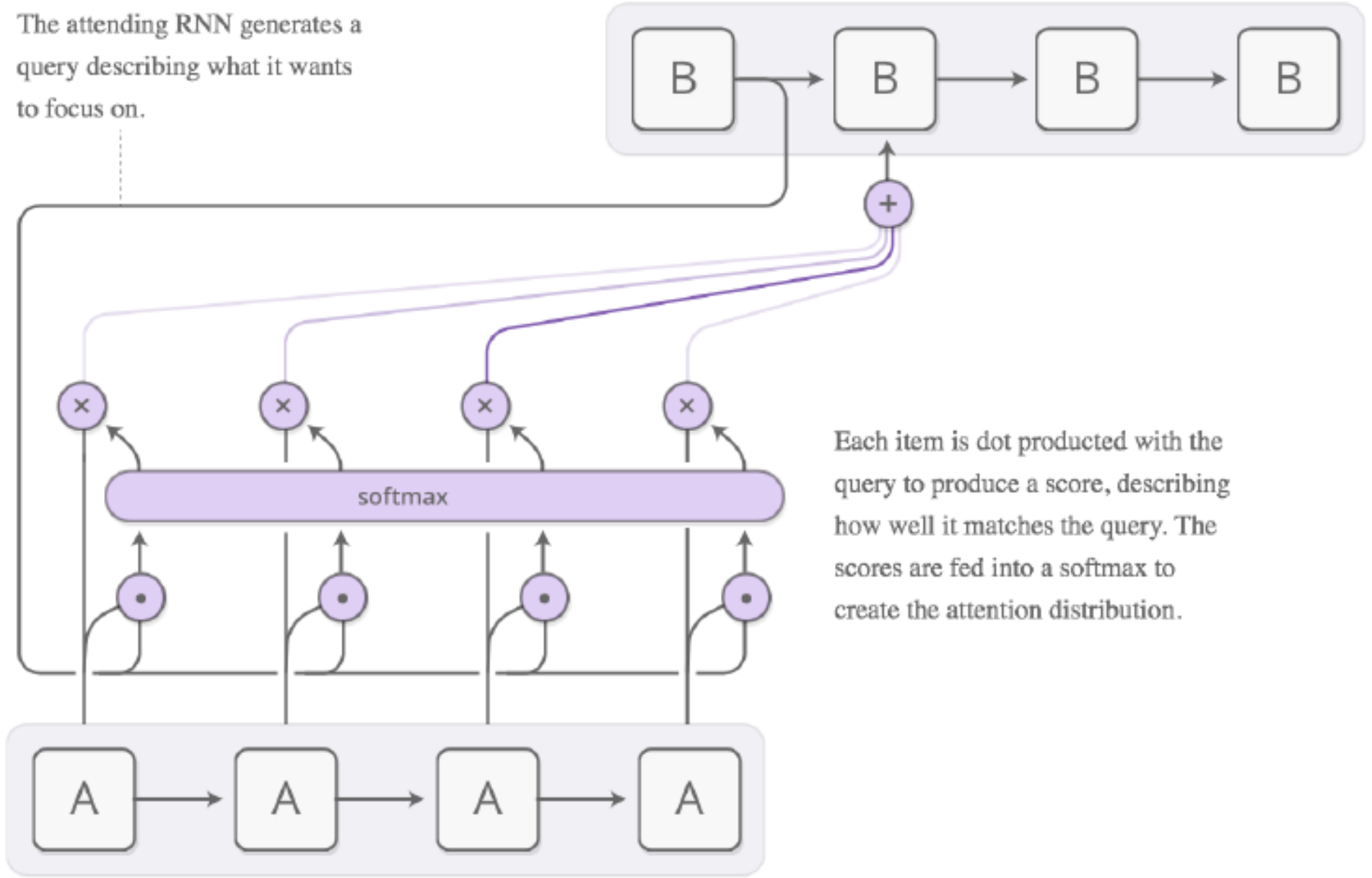
- ☑ Motivation
- ☐ **Graph Neural Network**
  - GCN
  - GraphSAGE
  - **GAT**
  - Gated GNN
- ☐ Applications

Aside: Attention

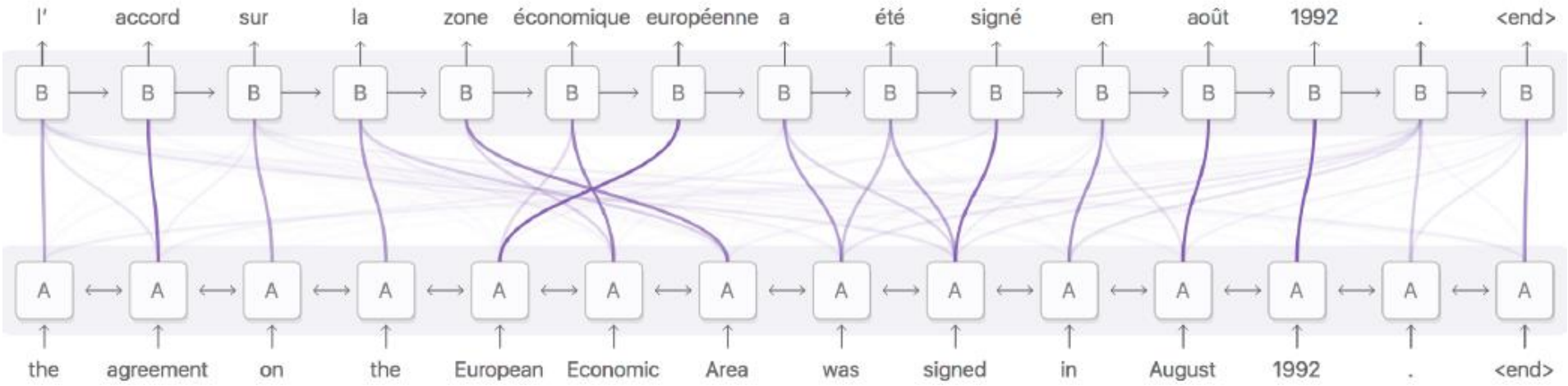
Attention

- Attention mechanism
  - De facto standard for sequential processing tasks
  - Computes linear combinations of the input features to generate the output. The coefficients of these linear combinations are parametrized by a shared neural network
  - Intuitively, allows each component of the output to generate its own combination of the inputs—thus, different outputs pay different levels of attention to the respective inputs.

Attention


The attending RNN generates a query describing what it wants to focus on.
B
B
B
B
+
softmax
A
A
A
A
Each item is dot producted with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.

Attention in Machine Translation



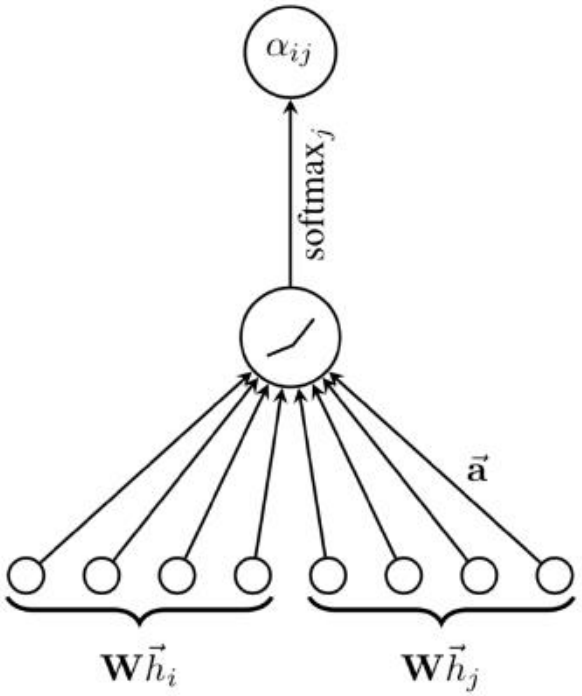
Self-attention

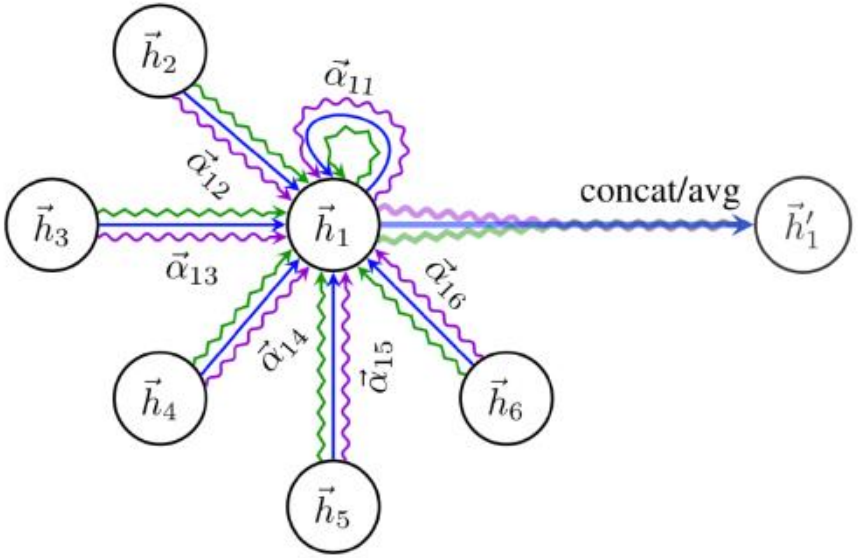
- A recent development on attention; the input attends over itself
  - $\alpha_{ij} = a(h_i, h_j)$
  - $h'_i = \sum_j softmax_j(\alpha_{ij}) h_j$
- Critically, this is parallelizable across all input positions
- Vaswani et al. (NIPS '17) have successfully demonstrated that this operation is self-sufficient for achieving state-of-the-art on machine translation

End of Aside

[Velickovic et al. (ICLR 2018)]

GNNs with Attention (GAT)





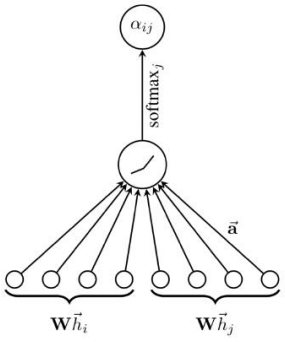
$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

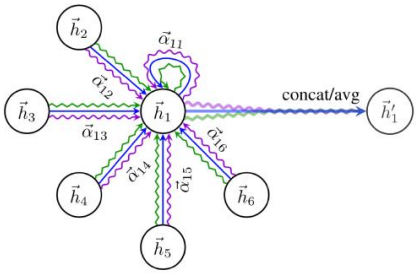
$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)}$$

[Velickovic et al. (ICLR 2018)]

GNNs with Attention (GAT)

- Pros
  - Effective for inductive setting
  - Slower than GCNs but faster than GNNs with edge embeddings
- Cons
  - (most likely) less expressive than GNNs with edge embeddings
  - Can be more difficult to optimize





Performance of GAT

	Transductive			*Inductive*
	Cora	**Citeseer**	**Pubmed**	**PPI**
# Nodes	2708	3327	19717	56944 (24 graphs)
# Edges	5429	4732	44338	818716
# Features/Node	1433	3703	500	50
# Classes	7	6	3	121 (multilabel)
# Training Nodes	140	120	60	44906 (20 graphs)
# Validation Nodes	500	500	500	6514 (2 graphs)
# Test Nodes	1000	1000	1000	5524 (2 graphs)

Performance of GAT

Transductive

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg	59.5%	60.1%	70.7%
SemiEmb	59.0%	59.6%	71.7%
LP	68.0%	45.3%	63.0%
DeepWalk	67.2%	43.2%	65.3%
ICA	75.1%	69.1%	73.9%
Planetoid	75.7%	64.7%	77.2%
Chebyshev	81.2%	69.8%	74.4%
GCN	81.5%	70.3%	**79.0%**
MoNet	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

Performance of GAT

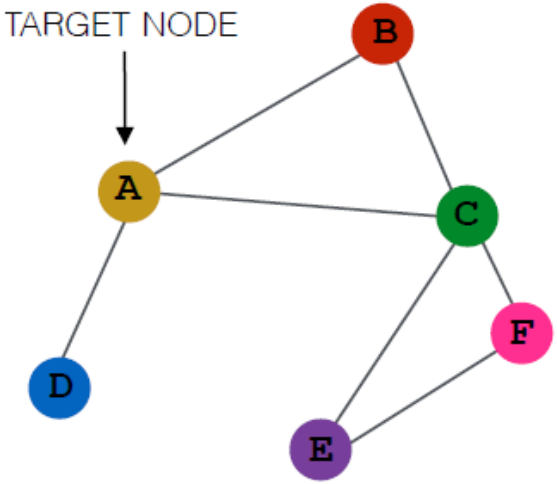
Inductive	
Method	**PPI**
Random	0.396
MLP	0.422
GraphSAGE-GCN	0.500
GraphSAGE-mean	0.598
GraphSAGE-LSTM	0.612
GraphSAGE-pool	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	$\mathbf{0.973} \pm 0.002$

Outline

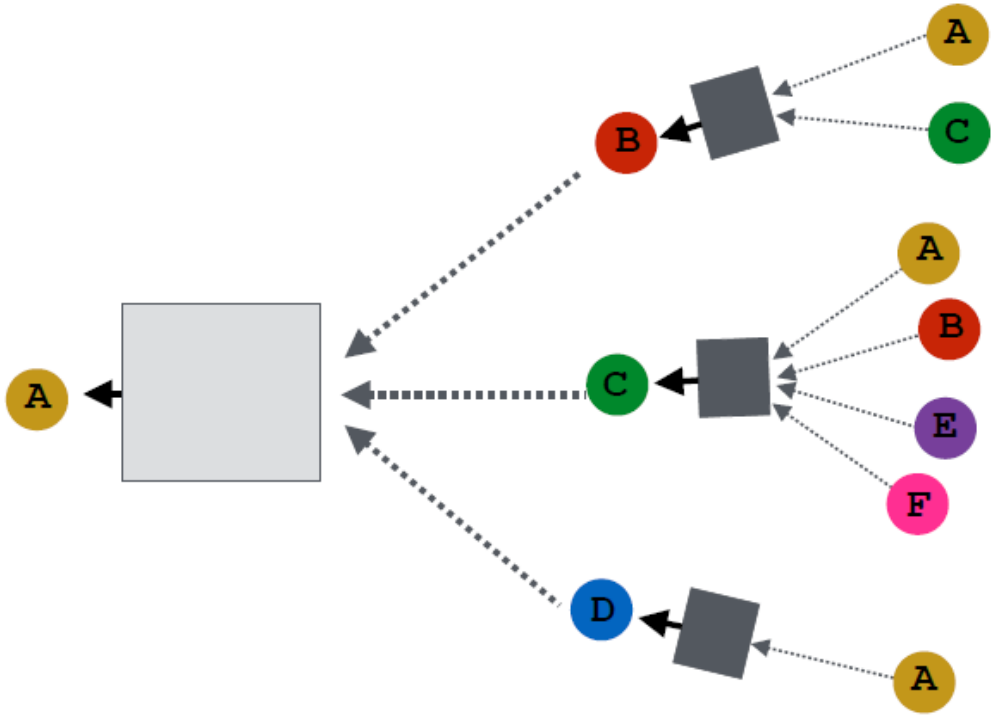
- ☑ Motivation
- ☐ **Graph Neural Network**
  - GCN
  - GraphSAGE
  - GAT
  - **Gated GNN**
- ☐ Applications

Neighborhood Aggreation

- Basic idea: nodes aggregate messages from their neighbors using neural networks

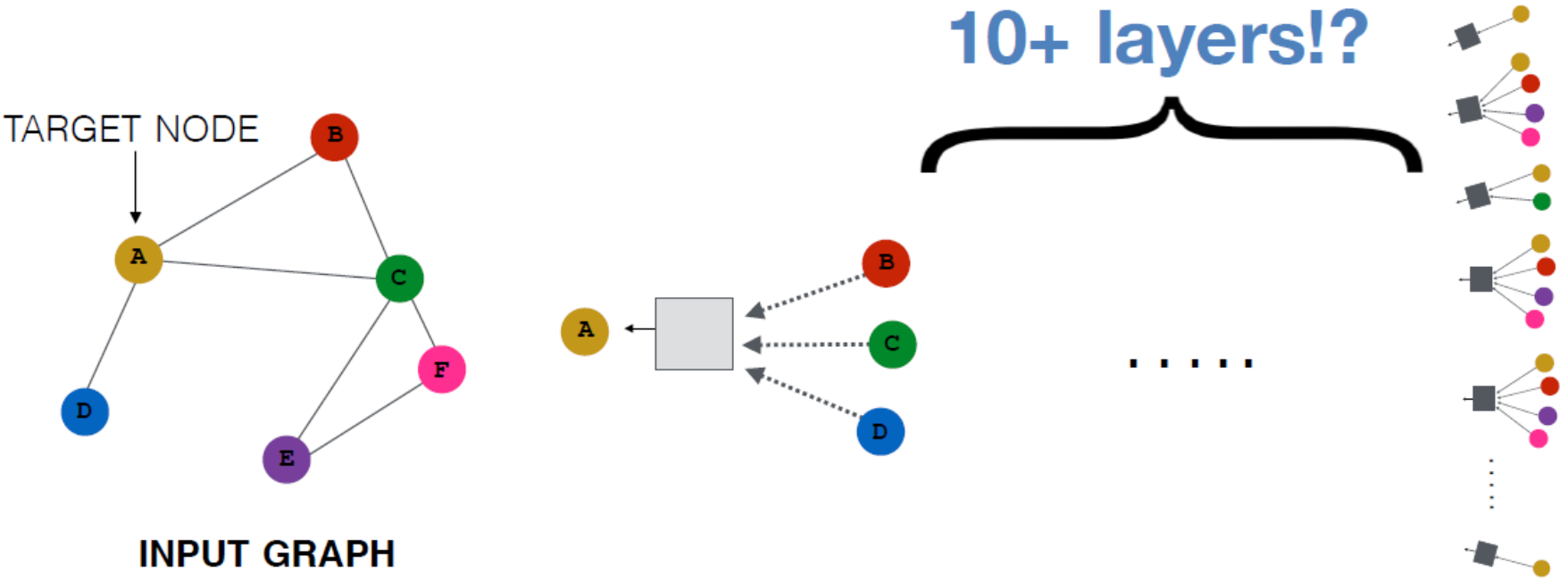


INPUT GRAPH



Neighborhood Aggreation

- GCNs and GraphSAGE work only for 2-3 layers; can we make a deep GNN?

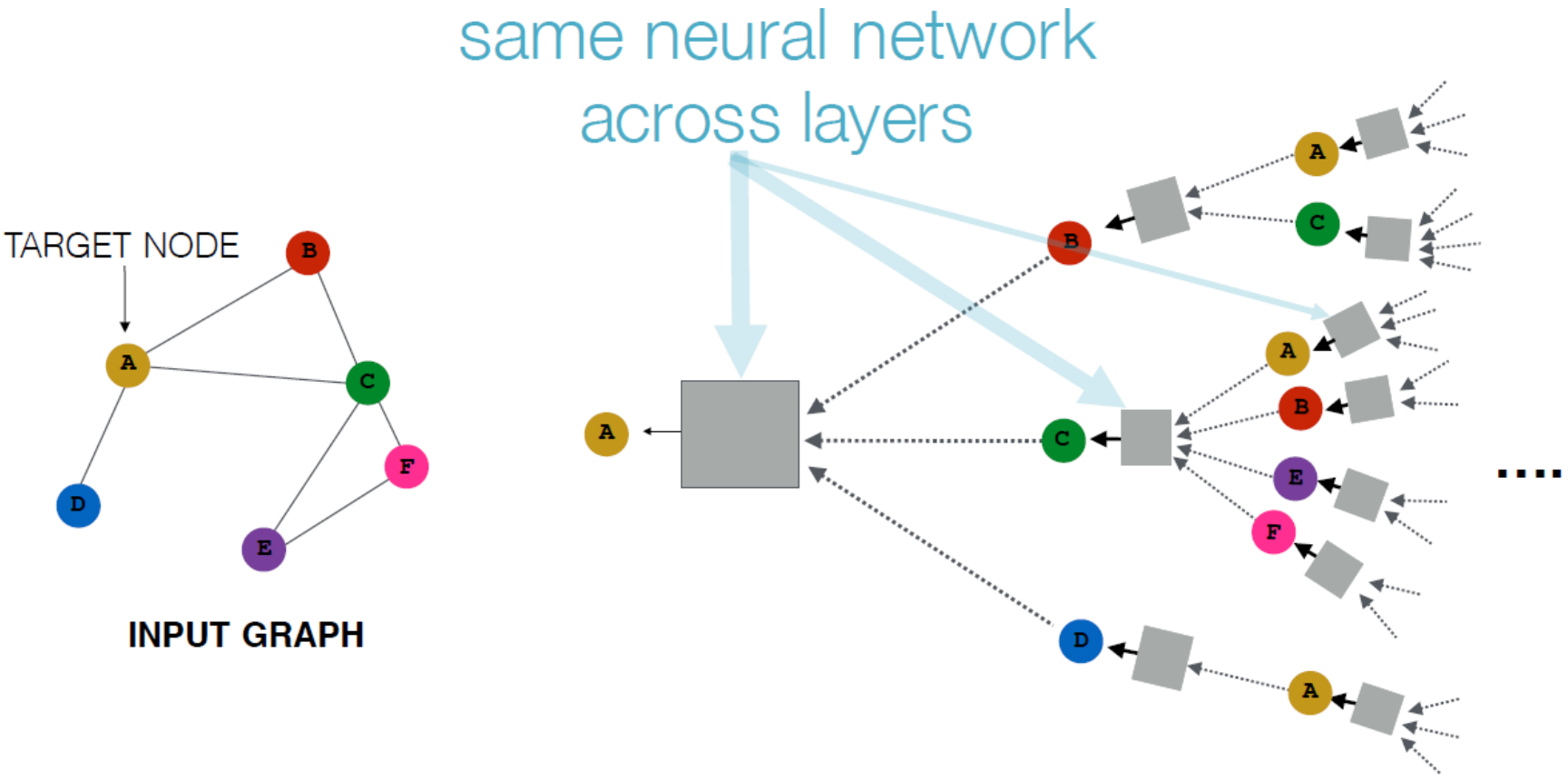


Gated GNN

- How can we build models with many layers of neighborhood aggregation?
- Challenges
  - Overfitting from too many parameters
  - Oversmoothing
  - Vanishing/exploding gradients during backpropagation
- Main idea
  - Exploit ideas from RNN

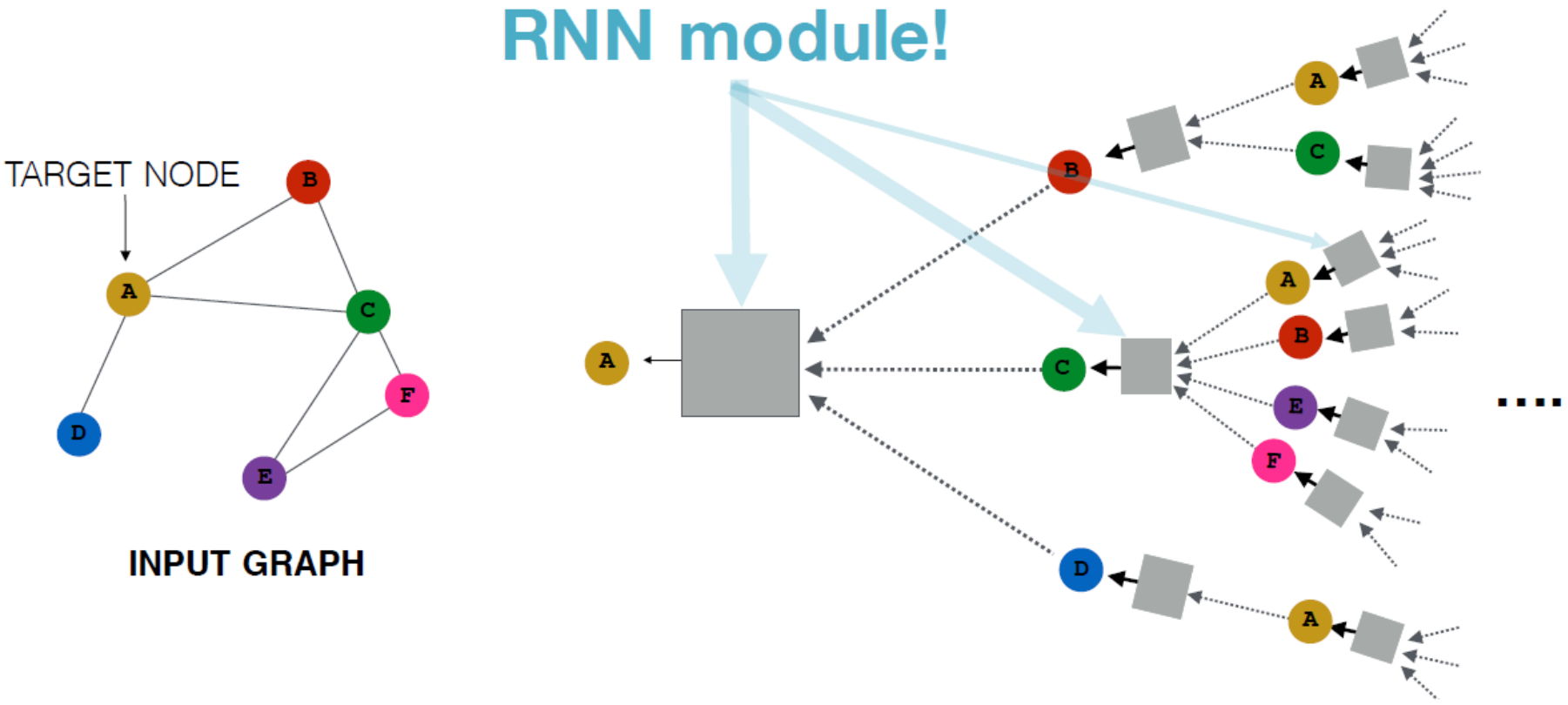
Gated GNN

- Main idea 1 : parameter sharing across layers



Gated GNN

- Main idea 2 : recurrent state update



Gated GNN

- Neighborhood aggregation with RNN state update
  - Get message from neighbors at step k

$$\mathbf{m}_v^k = \mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}$$

aggregation function does not depend on k

  - Update node state using GRU (Gated Recurrent Unit). New node state depends on the old state and the message from neighbors

$$\mathbf{h}_v^k = \mathrm{GRU}(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

Gated GNN

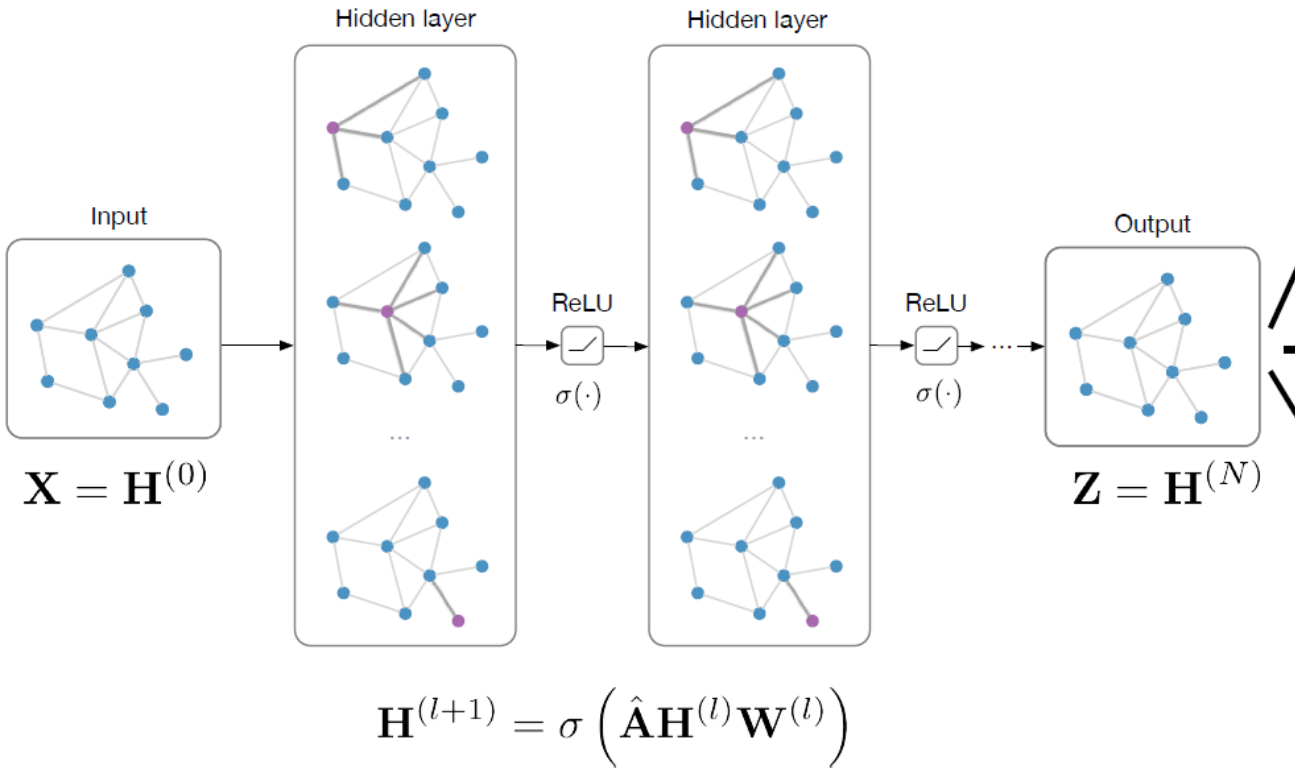
- Advantages
  - Can handle models with more than 20 layers
  - Most real-world networks have small diameters (e.g., < 7)
  - Allows for complex information about global graph structure to be propagated to all nodes

Outline

- ☑ Motivation
- ☑ Graph Neural Network
- ☐ **Applications**
  - **Node Classification**
  - Recommendation

Classification and Link Prediction with GNNs/GCNs

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



Node classification:

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

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

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

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

Link prediction:

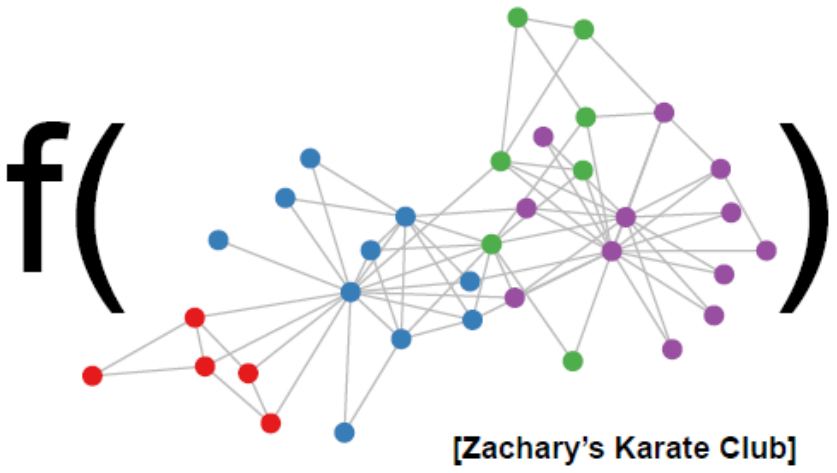
$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$

Kipf & Welling (NIPS BDL 2016)

What Do Learned Representations Look Like?

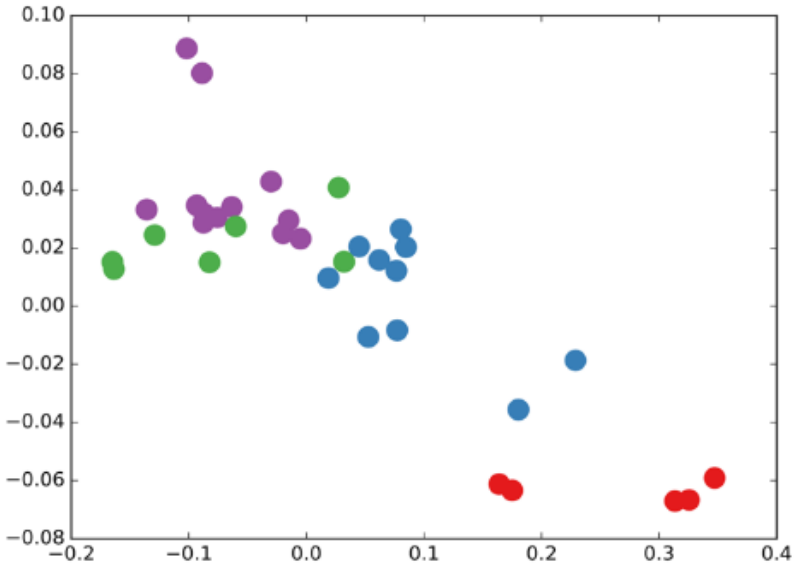
- Forward pass through **untrained** 3-layer GCN model

Parameters initialized randomly



=

2-dim output per node

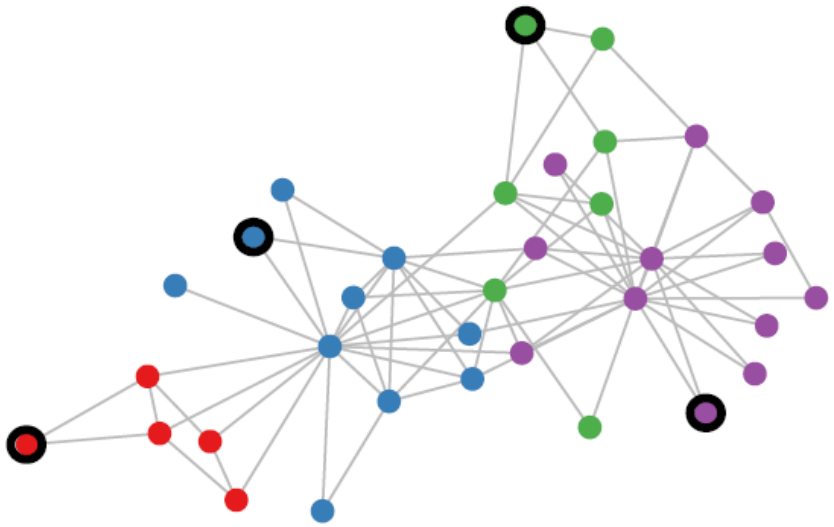


Semi-Supervised Classification on Graphs

Setting:

Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:
Predict node label of unlabeled nodes



Evaluate loss on labeled nodes only:

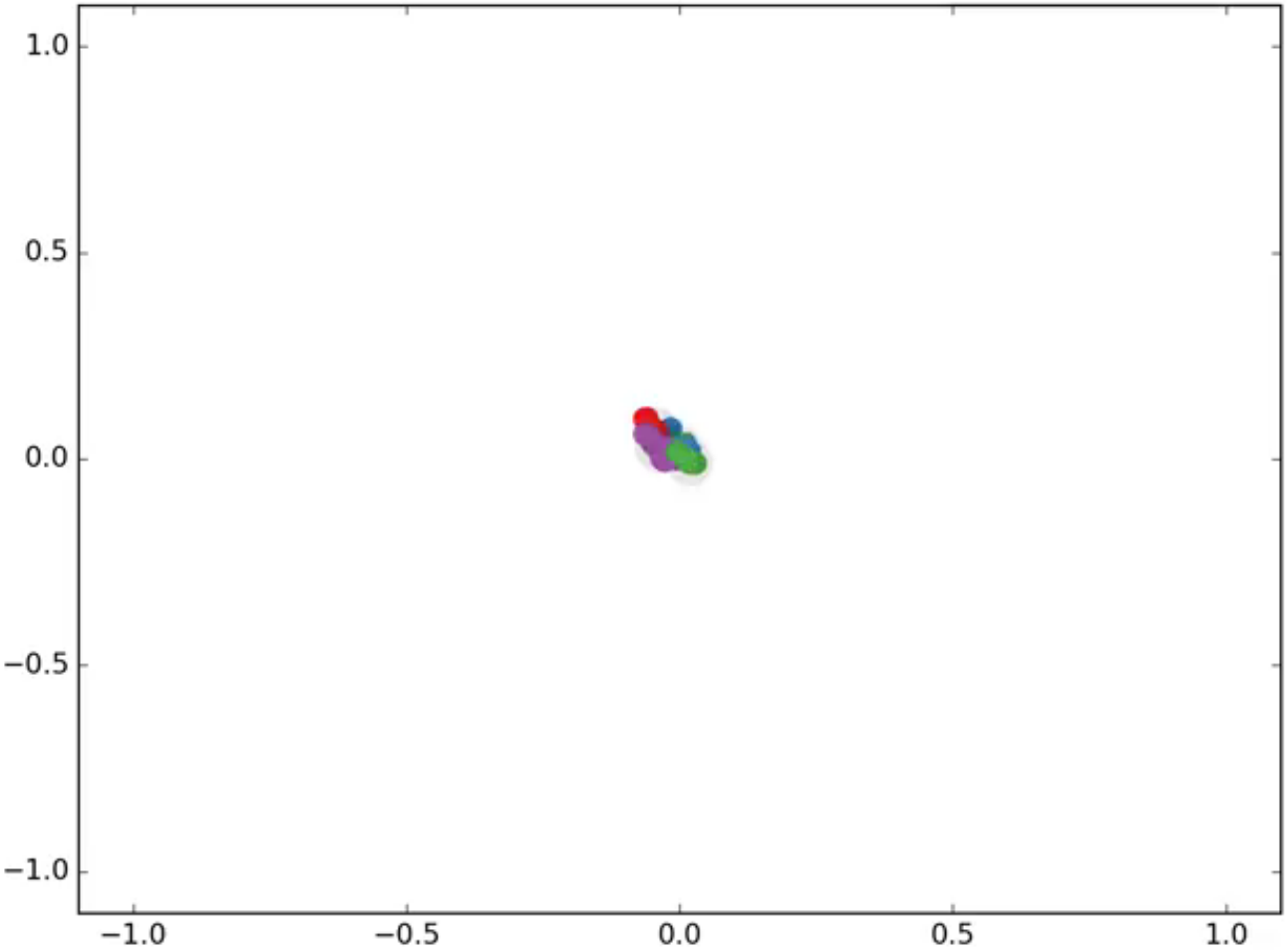
$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

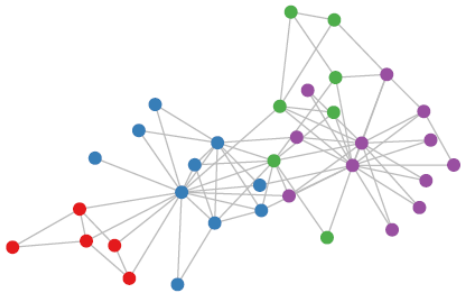
$\mathcal{Y}_L$ set of labeled node indices

$\mathbf{Y}$ label matrix

$\mathbf{Z}$ GCN output (after softmax)

Toy Example (semi-supervised learning)





Classification on Citation Networks

Input: Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

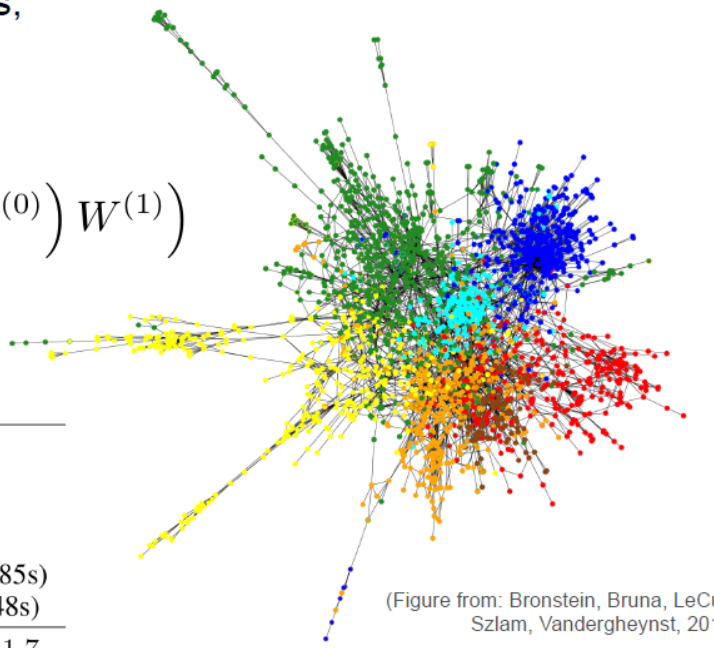
Target: Paper category (e.g. stat.ML, cs.LG, …)

Model: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\,W^{(1)}\Big)$

Classification results (accuracy)

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [24]	59.6	59.0	71.1	26.7
LP [27]	45.3	68.0	63.0	26.5
DeepWalk [18]	43.2	67.2	65.3	58.1
Planetoid* [25]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

no input features



(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

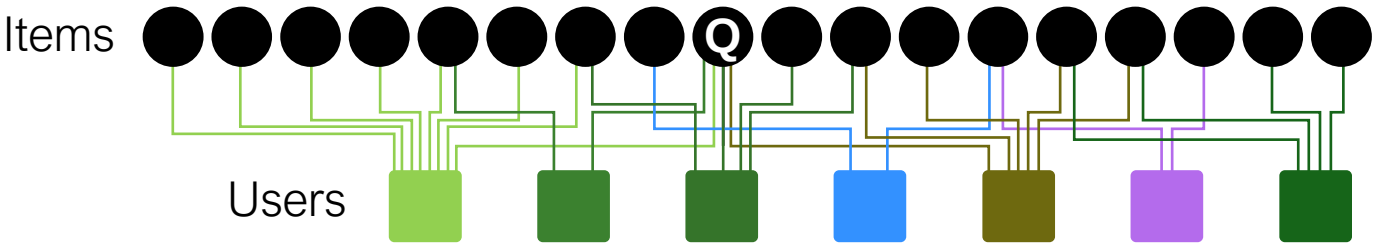
Outline

- ☑ Motivation
- ☑ Graph Neural Network
- ➡ ☐ **Applications**
  - Node Classification
  - ➡ **Recommendation**

Ying et al., Graph Convolutional Neural Networks for Web-Scale Recommender Systems, KDD'18

Bipartite Graph for RecSys

- Graph is dynamic: need to apply to new nodes without model retraining
- Rich node features: content, image



GNN for RecSys

- Two sources of information in traditional recommender systems
  - Content features: user and item features in the form of images, categories etc.
  - Network structure: user-item interactions, in the form of graph/network structure
- Graph neural networks (GNN) naturally incorporate both!

Application: Pinterest

Human curated collection of pins

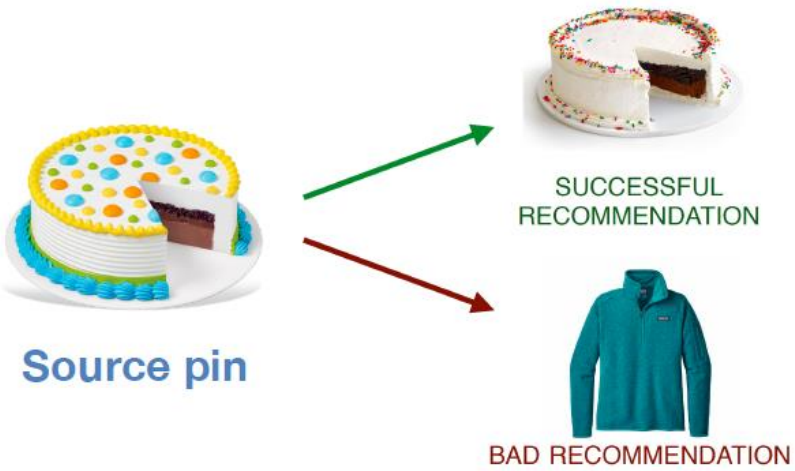
Pins: Visual bookmarks someone has saved from the internet to a board they've created.

Pin features: Image, text, link

Boards

Application: Pinterest

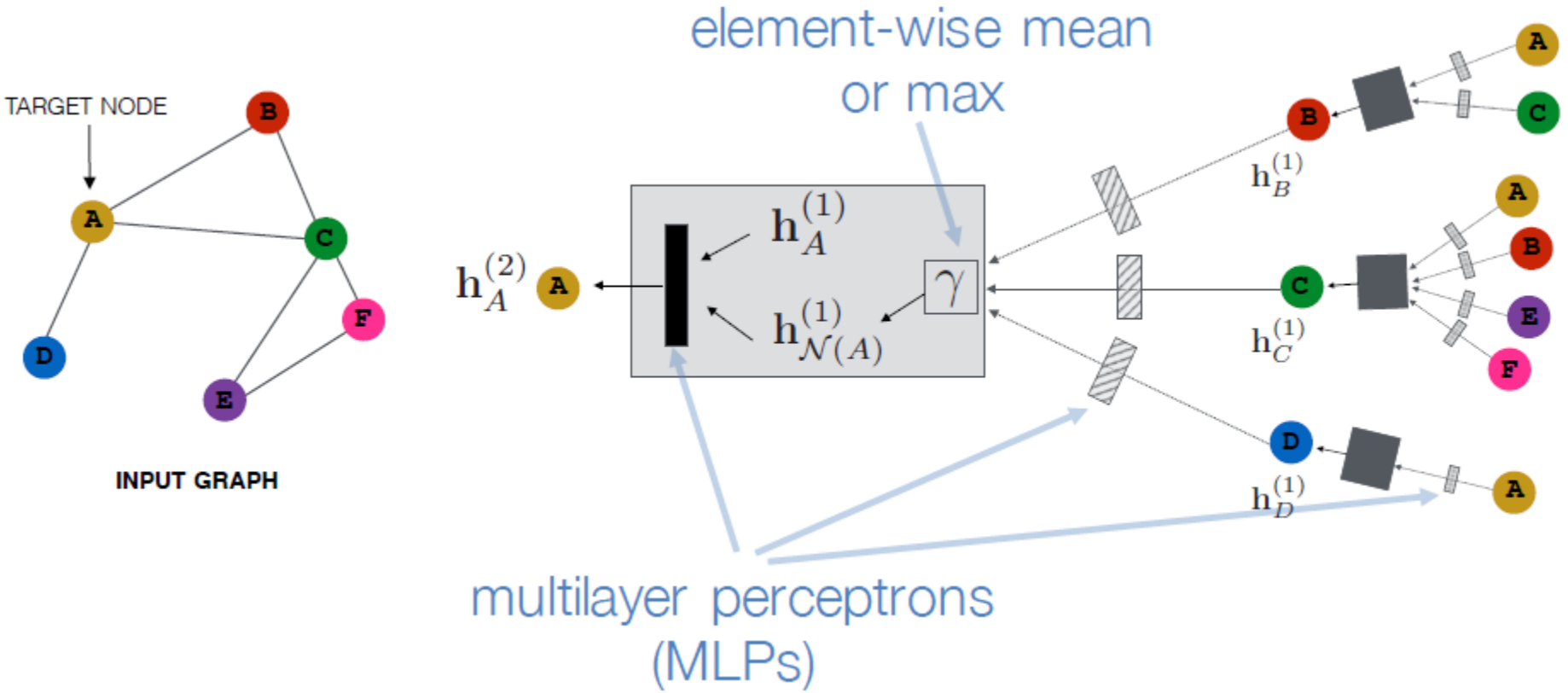
- Task: recommend related pins to users



- Challenges
  - Massive size: 3 billion pins and boards, 16 billion interactions
  - Heterogeneous data: rich image and text features

RW-GCN Overview

- RW-GCN = Random-Walk GCN
- Extension of GraphSAGE

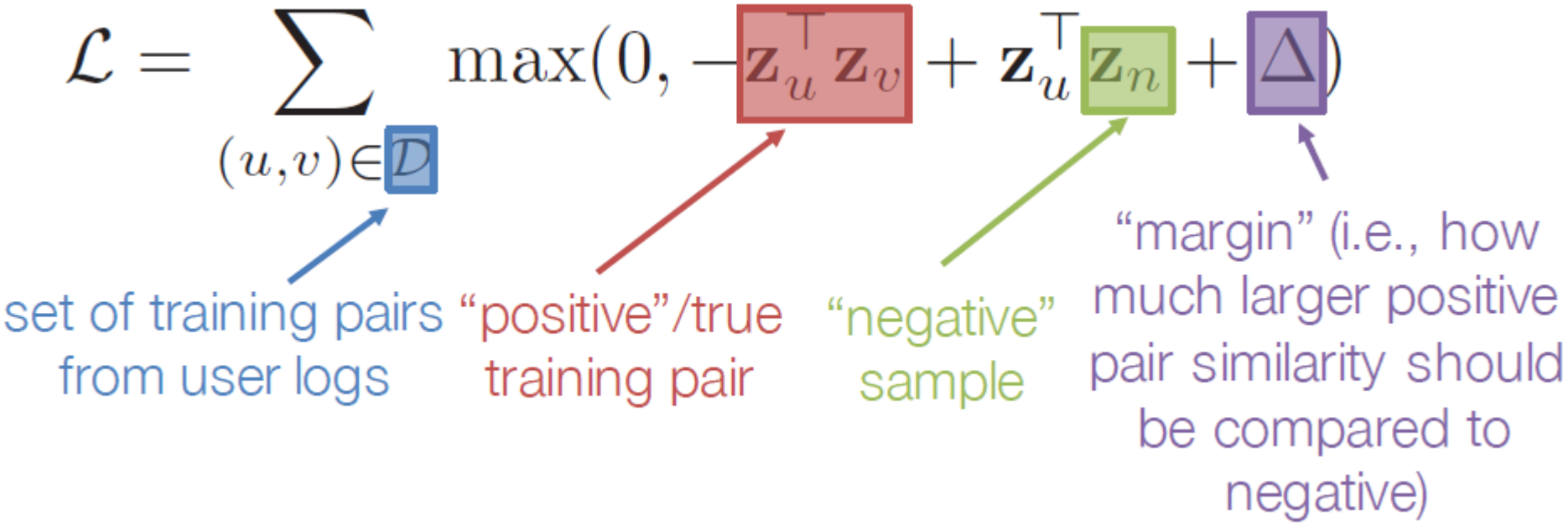


RW-GCN Pipeline

- Collect billions of training pairs from user logs.
- Train system to generate similar embeddings for training pairs.
- Generate embeddings for all pins.
- Make recommendations using nearest neighbor search in the embedding space (in real time).

Training RW-GCN

- Train so that pins that are consecutively clicked have similar embeddings
- Max-margin loss

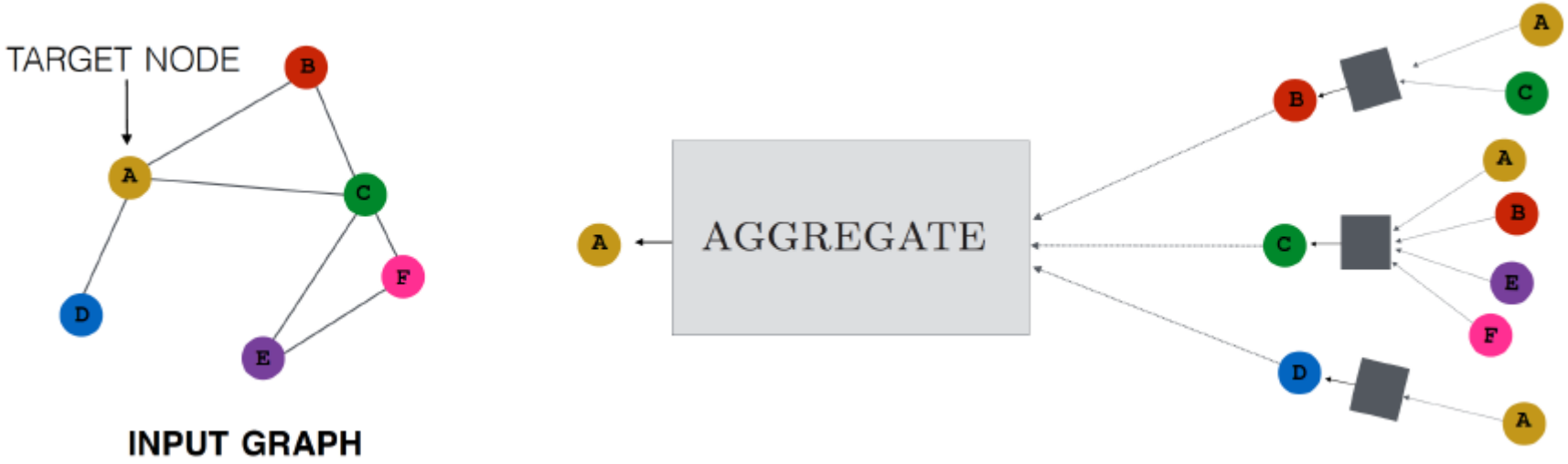


RW-GCN Efficiency

- 10,000X larger than any previous graph neural network application.

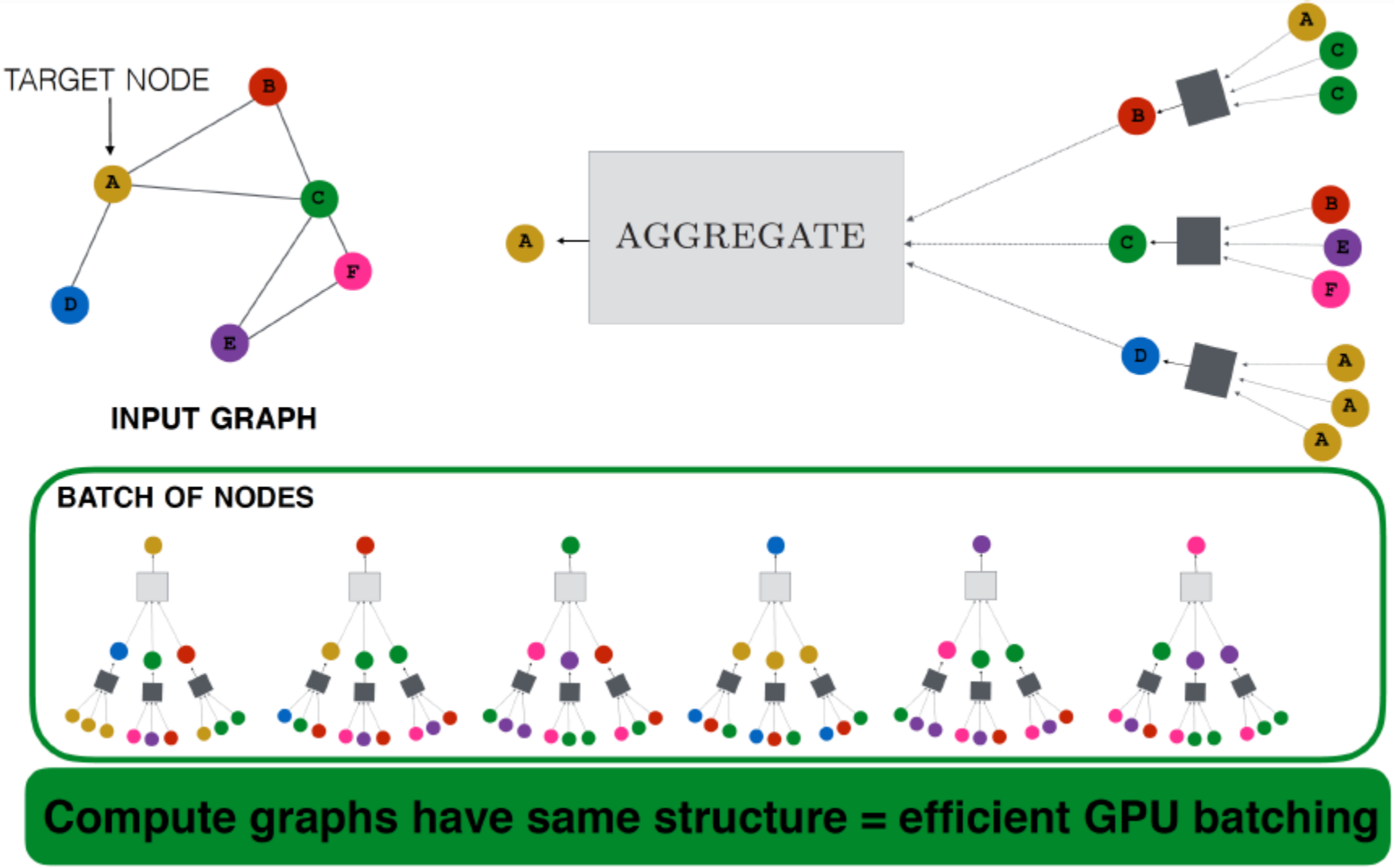
- Main ideas
  - Sub-sample neighborhoods for efficient GPU batching
  - Producer-consumer training pipeline
  - Curriculum learning for negative samples

Neighborhood Subsampling



Every node has unique compute graph. Can't batch on GPU!

Neighborhood Subsampling



Neighborhood Subsampling

- **Random-walk**-based neighborhood
  - Approximates personalized PageRank (PPR) score.
  - Sampled neighborhood for a node is a list of nodes with the top-K PPR score.

Producer-consumer Pipeline

- Select a batch of pins
- Run random walks
- Construct their computation graphs

CPU (producer)

- Multi-layer aggregations
- Loss computation
- Backprop

GPU (consumer)

Curriculum Learning

- Idea: use harder and harder negative samples
- Include more and more hard negative samples for each epoch

Source pin

Positive

Easy negative

Hard negative

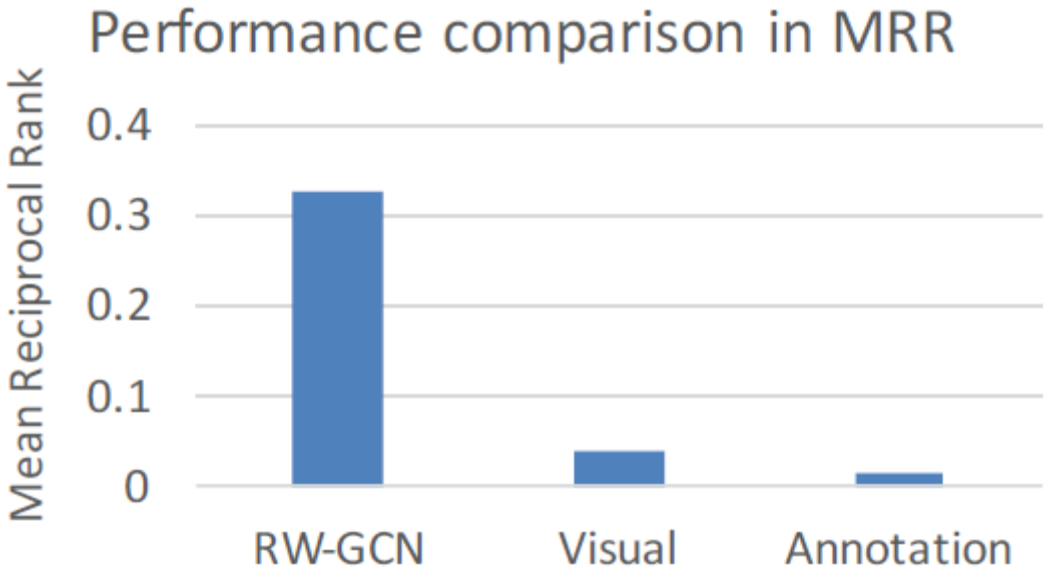
RW-GCN Performance

- 72% better recommendation quality than standard GraphSAGE model.

- Key contributions
  1. Weigh importance of neighbors according to approximate PPR score.
  2. Use curriculum training to provide harder and harder training examples over time.

RW-GCN Performance

- Set-up: rank true “next-clicked” pin against 10^9 other candidates



- MRR: mean reciprocal rank of true example
- Baseline: deep content-based models

What You Need to Know

- Motivation of graph deep learning
- Graph neural networks
- Applications of GNN

Questions?