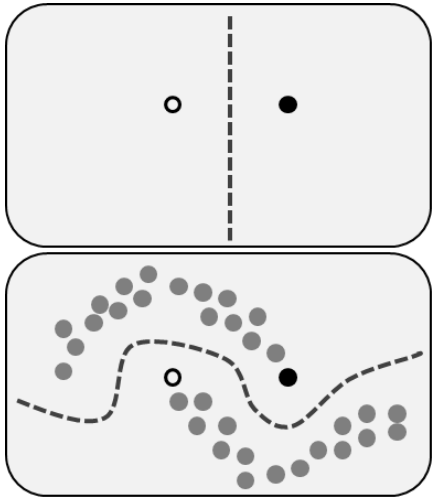
Label Propagation

Semi-supervised learning

- Given both labeled and unlabeled data
- Is unlabeled data useful?



Label Propagation

Semi-supervised learning

- Graph-based semi-supervised learning:Is unlabeled data useful?
  - Encode the unlabeled feature information into graph
- Two classical approaches:
  - Graph-based algorithm (label propagation)
  - Graph-based regularization (manifold regularization)

Label Propagation

Inductive vs Transductive

from existing graph

new graph

	Inductive (Generalize to unseen data)	Transductive (Doesn't generalize to unseen data)
Supervised (labeled data)	SVM, ...	X
Semi-supervised (labeled + unlabeled)	Manifold Regularization	Label propagation

Label Propagation

Main idea

- Smoothness Assumption:
  - If two data points are close to each other, their predictions should be similar
- Measure the similarity between data points (similarity graph)
- Enforce the predictions to be similar based on similarity graph

Label Propagation

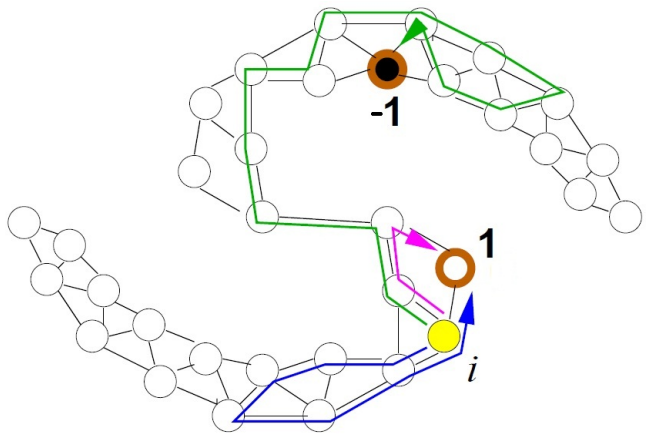
Similarity Graph

- Assume we have n data points $x_1, \ldots, x_n$
- Define the similarity matrix $S \in \mathbb{R}^{n \times n}$
  - $S_{ij} =$ similarity(x_i, x_j)
- The similarity function can be defined by many ways, for example,
  - similarity$(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
- S is a dense $n \times n$ matrix $\Rightarrow$ High computational cost
- Usually, a k-nearest neighbors graph is used:
  - $S_{ij} \neq 0$ only when j is in i's k-nearest neighbors

Label Propagation

Transductive setting using label propagation

- Input:
  - ℓ labeled training samples $x_1, \ldots, x_\ell$ with labels $y_1, \ldots, y_\ell$ (each y_i is a k dimensional label vector for multiclass/multilabel problems)
  - u unlabeled training samples $x_{\ell+1}, \ldots, x_{\ell+u}$
- Output: labels $y_{\ell+1}, \ldots, y_{\ell+u}$ for all unlabeled samples
- Main idea: propagate labels through the similarity matrix



Label Propagation

Label Propagation

- Proposed in Zhu et al., ``Semi-supervised Learning using Gaussian Fields and Harmonic Functions''. In ICML 2003.
- $T \in \mathbb{R}^{(\ell+u)\times(\ell+u)}$: transition matrix such that
  - $T_{ij} = P(j \to i) = \frac{S_{ij}}{\sum_{k=1}^{\ell+u} S_{kj}}$
- $Y \in \mathbb{R}^{(\ell+u)\times k}$: the label matrix. The first ℓ rows are labels $y_1, \ldots, y_\ell$. The rest u rows are initialized by 0 (will not affect the results).
- The Algorithm:
  - Repeat the following steps until convergence:
  - Step 1: Propagate labels by the transition matrix: $Y \leftarrow TY$
  - Step 2: Normalize each row of Y
  - Step 3: Reset the first ℓ rows of Y to be $y_1, \ldots, y_\ell$

Label Propagation

Label Propagation (convergence)

- The algorithm will converge to a simple solution!
- Step 1 and step 2 can be combined into
  - $Y \leftarrow \bar{T}Y$,
  - where $\bar{T}$ is the row-normalized matrix of T
- We focus on row $\ell + 1$ to $\ell + u$ (defined by Y_U)
  - $$\begin{pmatrix} Y_L \\ Y_U \end{pmatrix} \leftarrow \begin{pmatrix} \bar{T}_{\ell\ell} & \bar{T}_{\ell u} \\ \bar{T}_{u\ell} & \bar{T}_{uu} \end{pmatrix} \begin{pmatrix} Y_L \\ Y_U \end{pmatrix}$$

Label Propagation

$L = BB^T$

Graph Laplacian

- Graph Laplacian:
  - $L = D - S$, where D is a diagonal matrix with $D_{ii} = \sum_j S_{ij}$
- L is positive semi-definite (if S is nonnegative)
- Main property: for any vector z,
  - $z^T L z = \sum_{i,j} S_{ij}(z_i - z_j)^2$
- Measure the non-smoothness of z according to the similarity matrix S

Label Propagation

Another form of label propagation

- Assume we only have one label (binary classification)
- Another equivalent form of label propagation:
  - $$\arg\min_{\hat{y}} \sum_{i,j} S_{ij}(\hat{y}_i - \hat{y}_j)^2 = \hat{y}^T L \hat{y} := f(\hat{y})$$
    $$\text{s.t. } \hat{y}_{1:\ell} = y$$
  - where $\hat{y} \in \mathbb{R}^{\ell+u}$ is the estimation of the labels.
- Optimal solution: $\nabla_{\hat{y}_{\ell+1:u}} f(\hat{y}) = 0$
- $$\Rightarrow \begin{pmatrix} D_{\ell\ell} - S_{\ell\ell} & -S_{\ell u} \\ -S_{u\ell} & D_{uu} - S_{uu} \end{pmatrix} \begin{pmatrix} \hat{y}_\ell \\ \hat{y}_u \end{pmatrix} = \begin{pmatrix} ? \\ 0 \end{pmatrix}$$
- $$\Rightarrow (D_{uu} - S_{uu})\hat{y}_u - S_{u\ell}\hat{y}_\ell = 0$$
- $$\Rightarrow \hat{y}_u = (D_{uu} - S_{uu})^{-1} S_{u\ell}\hat{y}_\ell$$
- $$\Rightarrow \hat{y}_u = (I - D_{uu}^{-1} S_{uu})^{-1} D_{uu}^{-1} S_{u\ell}\hat{y}_\ell$$
- $$\Rightarrow \hat{y}_u = (I - \bar{T}_{uu})^{-1} \bar{T}_{u\ell}\hat{y}_\ell$$

Label Propagation

Experimental Results (Zhu et al., 2003)

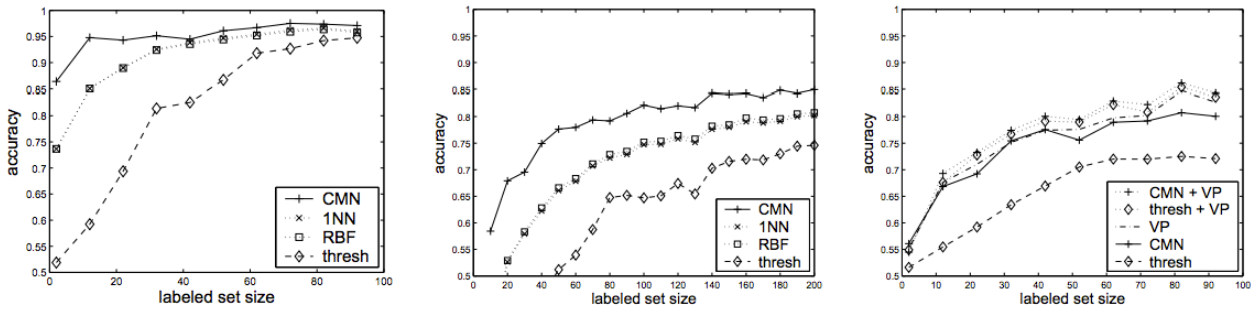


Figure 3. Harmonic energy minimization on digits "1" vs. "2" (left) and on all 10 digits (middle) and combining voted-perceptron with harmonic energy minimization on odd vs. even digits (right)

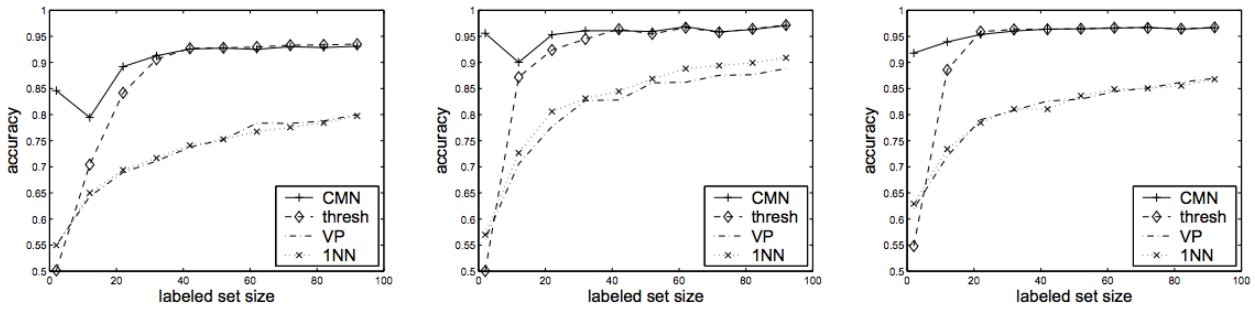
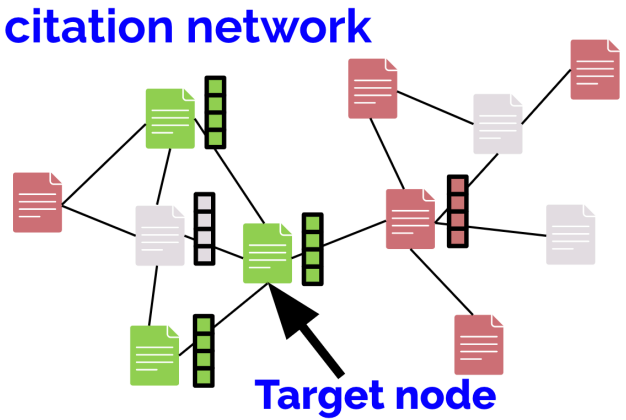


Figure 4. Harmonic energy minimization on PC vs. MAC (left), baseball vs. hockey (middle), and MS-Windows vs. MAC (right)

Graph Convolutional Neural Network

Node classification problem

- Given a graph of N nodes, with adjacency matrix $A \in \mathbb{R}^{N \times N}$
- Each node is associated with a D-dimensional feature vector.
- $X \in \mathbb{R}^{N \times D}$: each row corresponds to the feature vector of a node
- Observe labels for a subset of nodes: $Y \in \mathbb{R}^{N \times L}$, only observe a subset of rows, denoted by Y_S
- Goal: Predict labels for unlabeled nodes (transductive setting) or
- test nodes (inductive setting) or test graphs (inductive setting)



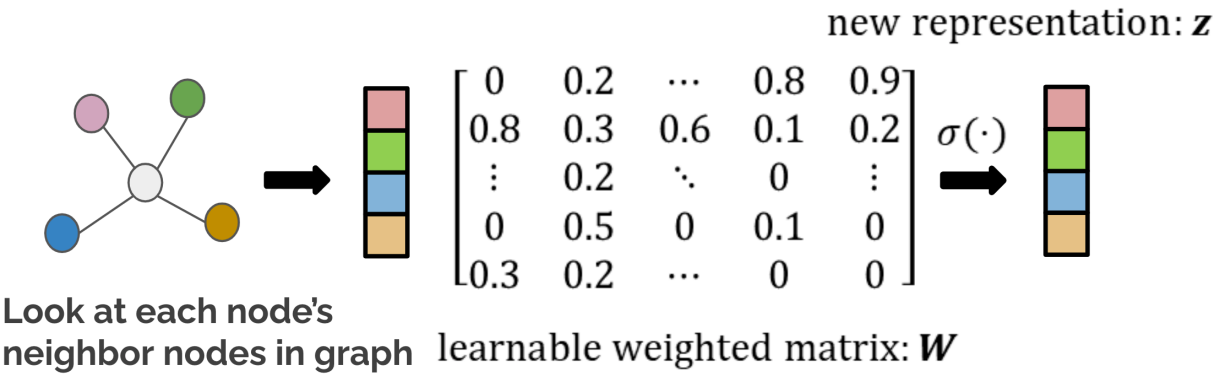
Graph Convolutional Neural Network

Graph Convolution Layer

- GCN: multiple graph convolution layers
- $\hat{A}$: normalized version of A:
  - $\tilde{A} = A + I, \quad \tilde{D}_{uv} = \sum_{v} \tilde{A}_{uv}, \quad P = \tilde{D}^{-1}\hat{A}$
- Graph convolution:
  - Input: features for each node $H^{(l)} \in \mathbb{R}^{n \times D}$
  - Output: features for each node $H^{(l+1)}$ after gathering neighborhood information
  - Convolution: $PH^{(l)}$: Aggregate features from neighbors
  - Convolution + fully-connected layer + nonlinear activation:
    - $H^{(l+1)} = \sigma(PH^{(l)}W^{(l)})$,
    - $W^{(l)}$ is the weights for the linear layer
    - $\sigma(\cdot)$: usually ReLU function

Graph Convolutional Neural Network

Graph convolutional network



Graph Convolutional Neural Network

Graph convolutional network

- Initial features $H^{(0)} := X$
- For layer $l = 0, \ldots, L$
  - $Z^{(l+1)} = PH^{(l)}W^{(l)}, \quad H^{(l+1)} = \sigma(Z^{(l+1)}),$
- Use final layer feature $H^{(L)} \in \mathbb{R}^{N \times K}$ for classification:
  - Loss $= \dfrac{1}{|S|} \displaystyle\sum_{s \in S} \text{loss}(y_s, Z_s^{(L)})$
  - Each row of $Z_s^{(L)}$ corresponds to the output score for each label
  - Cross-entropy loss for classification

Graph Convolutional Neural Network

Graph convolutional network

- Model parameters: $W^{(1)}, \cdots, W^{(L)}$
- Can be used to
  - Predict unlabeled nodes in the training set
  - Predict testing nodes (not in the training set)
  - Predict labels for a new graph
- Also, features extracted by GCN $H^{(L)}$ is usually very useful for other tasks

Graph Convolutional Neural Network

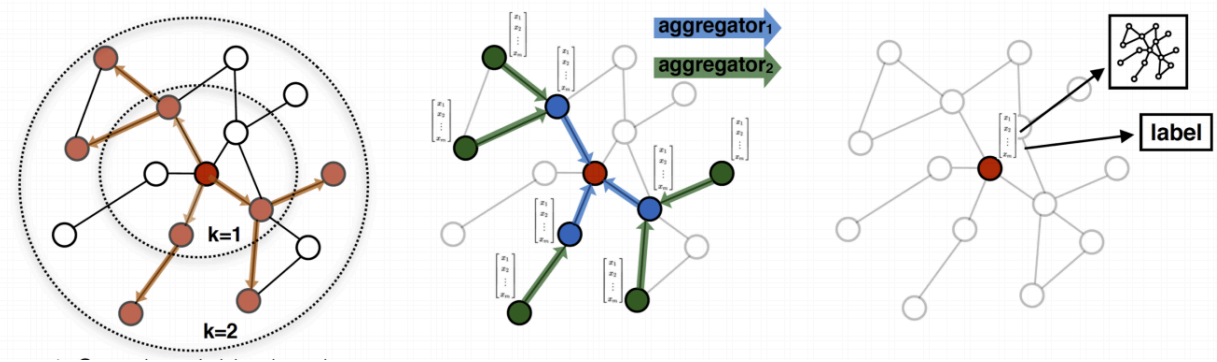
GCN training

- Full Gradient descent in the original paper (Kipf & Welling, 2017):
  - Need many iterations (epochs)
  - Large memory requirement: $O(NDL)$ for storing all the intermediate embeddings
- Can we use SGD/Adam?

Graph Convolutional Neural Network

Training GCN is non-trivial

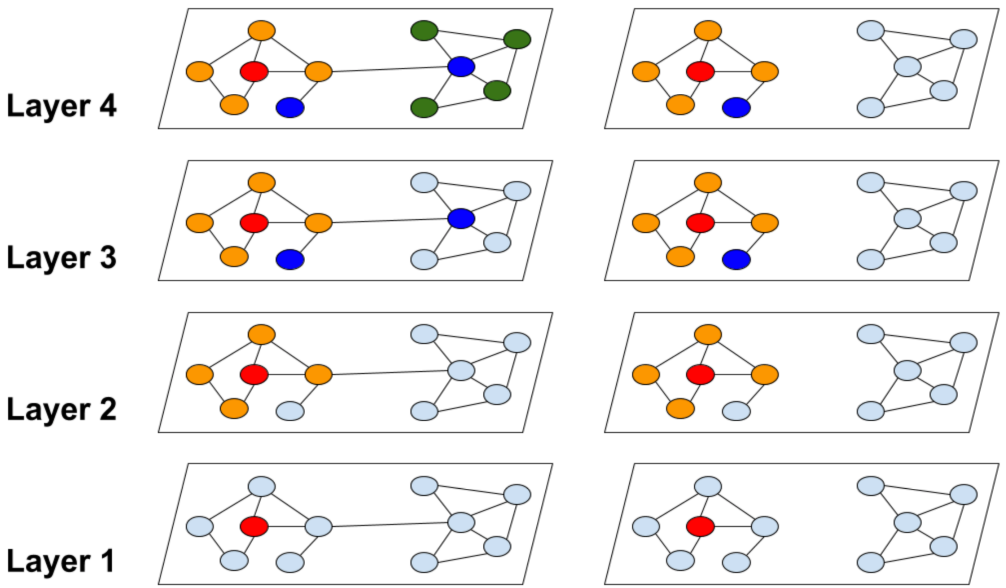
- Loss on a node depends on all its neighbors
- Neighborhood explosion issue during training
- This dependency brings both computation and memory challenges for training
- Previous idea: subsample a smaller number of neighbors
  - GraphSAGE (NeurIPS '17), VRGCN (ICML '18)



Graph Convolutional Neural Network

Graph-batching: more efficient training

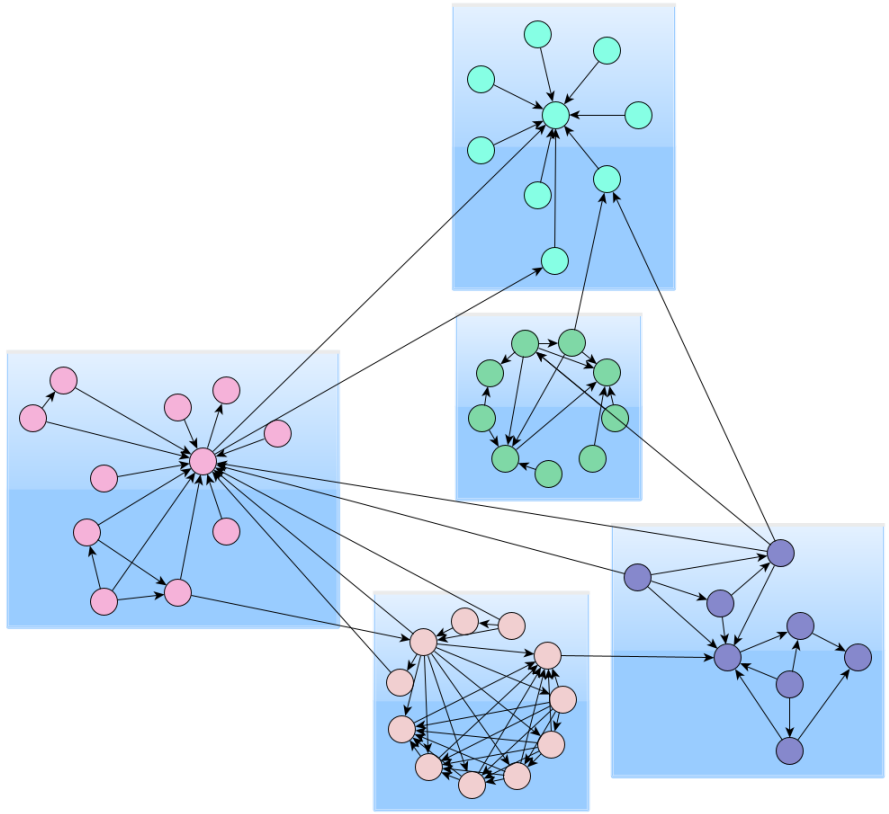
- Sample a batch of nodes at each time (B)
- For each SGD update, conduct forward and backward propagation only within B
- No neighborhood explosion anymore
- May lead to biased gradient estimator since some links are discarded in gradient computation



Graph Convolutional Neural Network

Cluster-GCN

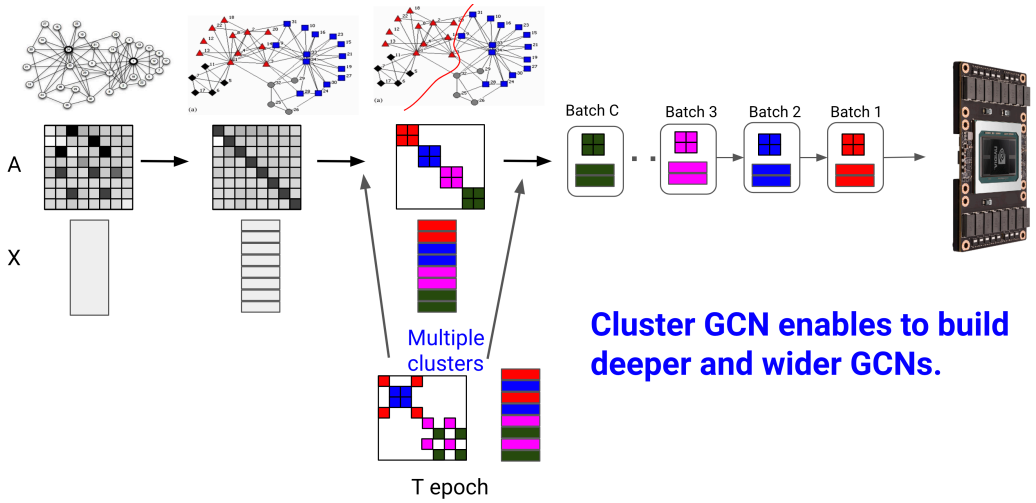
- Graph batching will be perfect if no between-batch links
- Find partition of the graph to minimize within-cluster links
- ⇒ graph clustering
- Efficient tools such as METIS can cluster million-scale graphs within few minutes.



Graph Convolutional Neural Network

Cluster-GCN (Cont'd)

- Cluster-GCN: run graph clustering, then each time sample a batch
- Equivalent to removing all the edges in off-diagonal blocks
- Multi-cluster trick:
  - Cluster into M cluster (with large M)
  - Each time form batch with k clusters
  - Each between-cluster edge has chances to be sampled



Graph Convolutional Neural Network

Results (Cluster-GCN)

- Enable training for deeper and wider GCN
  - PPI: 5-layer with 2048 hidden units
  - Reddit: 4-layer with 128 hidden units
- Used to achieve SoTA results

Table 8: Comparisons of running time, memory and testing accuracy (F1 score) for Amazon2M.

	Time		Memory		Test F1 score	
	VRGCN	Cluster-GCN	VRGCN	Cluster-GCN	VRGCN	Cluster-GCN
Amazon2M (2-layer)	337s	1223s	7476 MB	2228 MB	89.03	89.00
Amazon2M (3-layer)	1961s	1523s	11218 MB	2235 MB	90.21	90.21
Amazon2M (4-layer)	N/A	2289s	OOM	2241 MB	N/A	90.41

Table 10: State-of-the-art performance of testing accuracy reported in recent papers.

	PPI	Reddit
FastGCN [1]	N/A	93.7
GraphSAGE [5]	61.2	95.4
VR-GCN [2]	97.8	96.3
GaAN [16]	98.71	96.36
GAT [14]	97.3	N/A
GeniePath [10]	98.5	N/A
Cluster-GCN	**99.36**	**96.60**

Graph Convolutional Neural Network

GraphSaint

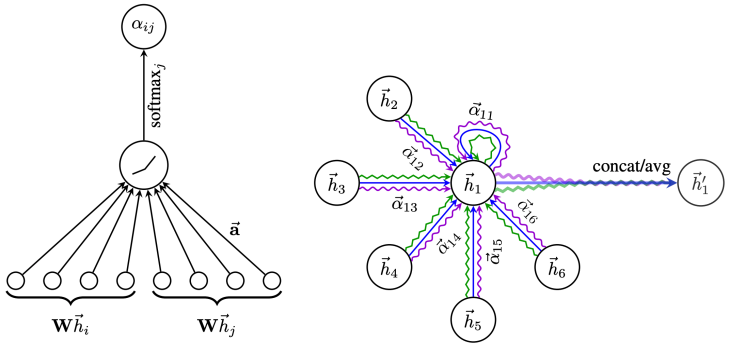
- A dynamic graph-batching method
- Each time form the batch (cluster) by
  - Random walk with restart (from a set of seed nodes)
  - Can be viewed as an algorithm to find a community with a given seed
- Better gradient estimator, but worse time complexity (due to dynamically finding a community)

	PPI (large version)		Reddit	
	2×512	5×2048	2×128	4×128
ClusterGCN	0.903±0.002	0.994±0.000	0.954±0.001	0.966±0.001
`GraphSAINT`	**0.941**±0.003	**0.995**±0.000	**0.966**±0.001	**0.970**±0.001

Graph Convolutional Neural Network

Graph Attention Networks

- Each edge may not contribute equally
- Using attention mechanism to automatically assign weights to each edge:
  - $\alpha_{i,j} = \frac{\exp(\text{LeakyReLU}(a^T[Wh_i \mid Wh_j]))}{\sum_{k \in N_i} \exp(\text{LeakyReLU}(a^T[Wh_i \mid Wh_k]))}$
    - where h_i, h_j are the features for node i and j at previous layer, W is the GNN weight, a is the additional learnable parameter for attention



Graph Convolutional Neural Network

SAGN and GAMLP

- Some simple approaches may also work for node prediction problems
- SAGN/GAMLP: Directly generate higher order graphs and conduct prediction, using attention to learn the weight of features

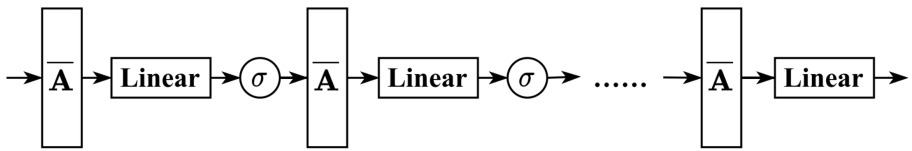


Figure 1: Architecture of common GNNs.

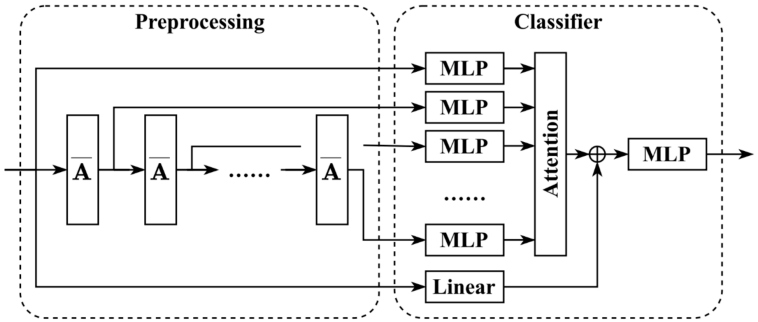


Figure 2: Architecture of SAGN. The multi-hop encoders and post encoder are depicted in form of MLP. The bottom "linear" refers to the residual linear layer. The symbol ⊕ represents the operation of summing integrated representation and residual term.

Graph Convolutional Neural Network

Correct and Smooth

- A post-processing method:
  - Propagate the residual the correct the prediction of the base model (e.g., MLP or GNN)
  - Label propagation to obtain the final output

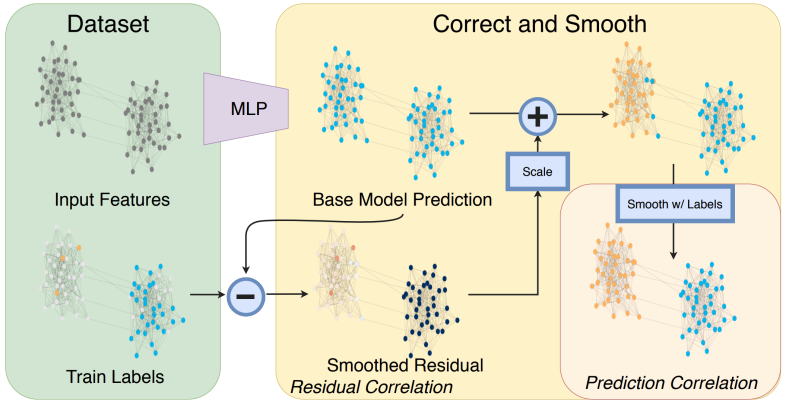


Figure 1: Overview of our GNN-free model, Correct and Smooth, with a toy example. The left cluster belongs to orange and the right cluster belongs to blue. We use MLPs for base predictions, ignoring the graph structure, which we assume gives the same prediction on all nodes in this example. After, base predictions are corrected by propagating errors from the training data. Finally, corrected predictions are smoothed with label propagation.

Graph Convolutional Neural Network

OGB Leaderboard: Node Classification

Scale	Name	Package	#Nodes	#Edges*	#Tasks	Split Type	Task Type	Metric
Medium	ogbn-products	>=1.1.1	2,449,029	61,859,140	1	Sales rank	Multi-class classification	Accuracy
Medium	ogbn-proteins	>=1.1.1	132,534	39,561,252	112	Species	Binary classification	ROC-AUC
Small	ogbn-arxiv	>=1.1.1	169,343	1,166,243	1	Time	Multi-class classification	Accuracy
Large	ogbn-papers100M	>=1.2.0	111,059,956	1,615,685,872	1	Time	Multi-class classification	Accuracy
Medium	ogbn-mag	>=1.2.1	1,939,743	21,111,007	1	Time	Multi-class classification	Accuracy

Rank	Method	Ext. data	Test Accuracy	Validation Accuracy	Contact	References	#Params	Hardware	Date
1	**GIANT-XRT+SAGN+MCR+C&S**	Yes	0.8673 ± 0.0008	0.9387 ± 0.0002	Yufei He (CogDL Team)	Paper, Code	1,154,654	GeForce RTX™ 3090 24GB (GPU)	Dec 8, 2021
2	**GIANT-XRT+SAGN+MCR**	Yes	0.8651 ± 0.0009	0.9389 ± 0.0002	Yufei He (CogDL Team)	Paper, Code	1,154,654	GeForce RTX™ 3090 24GB (GPU)	Dec 8, 2021
3	**GIANT-XRT+SAGN+SLE+C&S (use raw text)**	Yes	0.8643 ± 0.0020	0.9352 ± 0.0005	Eli Chien (UIUC)	Paper, Code	1,548,382	Tesla T4 (16GB GPU)	Nov 8, 2021
4	**GIANT-XRT+SAGN+SLE (use raw text)**	Yes	0.8622 ± 0.0022	0.9363 ± 0.0005	Eli Chien (UIUC)	Paper, Code	1,548,382	Tesla T4 (16GB GPU)	Nov 8, 2021
5	**GIANT-XRT+GAMLP+MCR**	Yes	0.8591 ± 0.0008	0.9402 ± 0.0004	Yufei He (CogDL Team)	Paper, Code	2,144,151	GeForce RTX™ 3090 24GB (GPU)	Dec 8, 2021
6	**GAMLP+RLU+SCR+C&S**	No	0.8520 ± 0.0008	0.9304 ± 0.0005	Yufei He (CogDL Team)	Paper, Code	3,335,831	GeForce RTX™ 3090 24GB (GPU)	Dec 8, 2021
7	**GAMLP+RLU+SCR**	No	0.8505 ± 0.0009	0.9292 ± 0.0005	Yufei He (CogDL Team)	Paper, Code	3,335,831	GeForce RTX™ 3090 24GB (GPU)	Dec 8, 2021
8	**SAGN+SLE (4 stages)+C&S**	No	0.8485 ± 0.0010	0.9302 ± 0.0003	Chuxiong Sun (CTRI)	Paper, Code	2,179,678	Tesla V100 (16GB GPU)	Sep 21, 2021
9	**SAGN+SLE (4 stages)**	No	0.8468 ± 0.0012	0.9309 ± 0.0007	Chuxiong Sun (CTRI)	Paper, Code	2,179,678	Tesla V100 (16GB GPU)	Sep 21, 2021