# FastGCN: Fast Learning with Graph Convolution Networks Via Importance Sampling

Jie Chen, Tengfei Ma, Cao Xiao
IBM Research

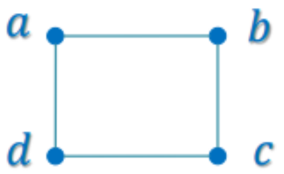
ICLR 2018

Sanghyeon Lee

26 Aug. 2020

# Backgrounds

### 1) Property of Adjacency Matrices ( A )



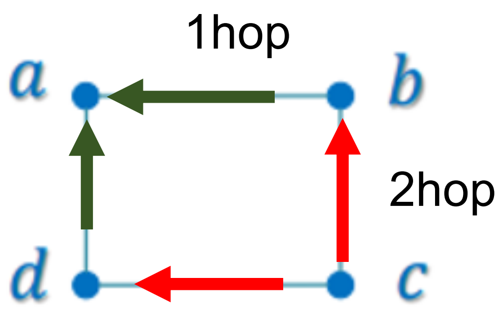

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

$$A^2 = AA = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \end{pmatrix}$$

$$A^3 = A^2A = \begin{pmatrix} 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \end{pmatrix}\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 4 & 0 & 4 \\ 4 & 0 & 4 & 0 \\ 0 & 4 & 0 & 4 \\ 4 & 0 & 4 & 0 \end{pmatrix}$$

### 2) Adjacency Matrices and Multi hop

- Aggregating feature from connected 1 hop node




$$Aggeratied\ node\ feature = AH$$

# Backgrounds

## 3) GCN

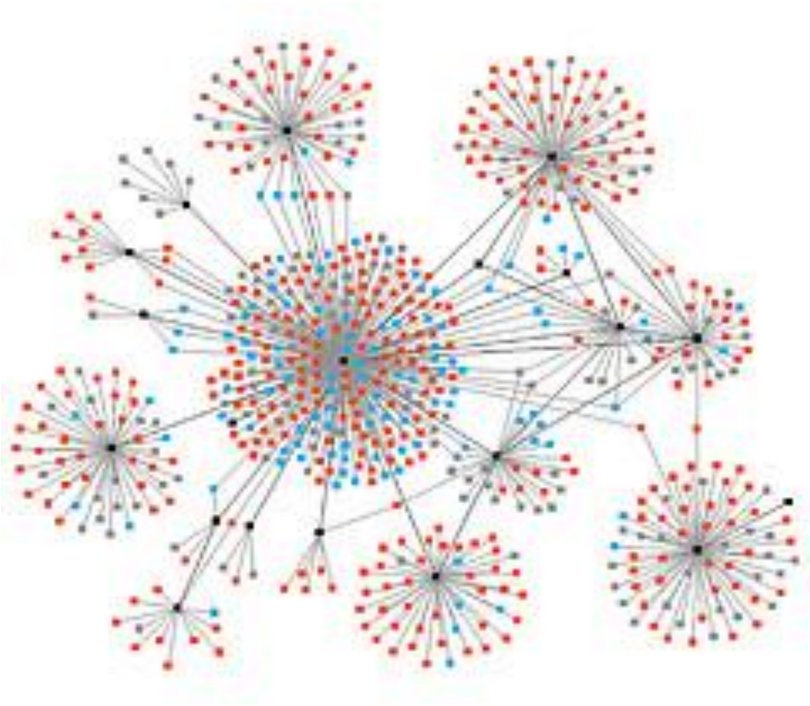
$$H_1^{(l+1)} = \sigma\left(H_1^{(l)}W^{(l)} + H_2^{(l)}W^{(l)} + H_3^{(l)}W^{(l)} + H_4^{(l)}W^{(l)} + b^{(l)}\right)$$

$$\Rightarrow \quad H_i^{(l+1)} = \sigma\left(\sum_{j\in N(i)} H_j^{(l)}W^{(l)} + b^{(l)}\right) \qquad H^{(l+1)} = \sigma\left(AH^{(l)}W^{(l)} + b^{(l)}\right)$$

$$\underset{H'}{\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & & & & & \\ 1 & & & & & \\ 2 & & & & & \\ 3 & & & & & \\ 4 & & & & & \end{array}} = \underset{A}{\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0 & 1 & 1 & 1 & 0 \\ 3 & 0 & 1 & 1 & 1 & 1 \\ 4 & 1 & 1 & 0 & 1 & 1 \end{array}} * \underset{H*W}{\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & & & & & \\ 1 & & & & & \\ 2 & & & & & \\ 3 & & & & & \\ 4 & & & & & \end{array}}$$

$$\underset{H*W}{\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & & & & & \\ 1 & & & & & \\ 2 & & & & & \\ 3 & & & & & \\ 4 & & & & & \end{array}} = \underset{H}{\begin{array}{c|cccccccccc} & F1 & F2 & F3 & F4 & F5 & F6 & F7 & F8 & F9 & F10 \\ \hline 0 & & & & & & & & & & \\ 1 & & & & & & & & & & \\ 2 & & & & & & & & & & \\ 3 & & & & & & & & & & \\ 4 & & & & & & & & & & \end{array}} * W$$

small-world **network**

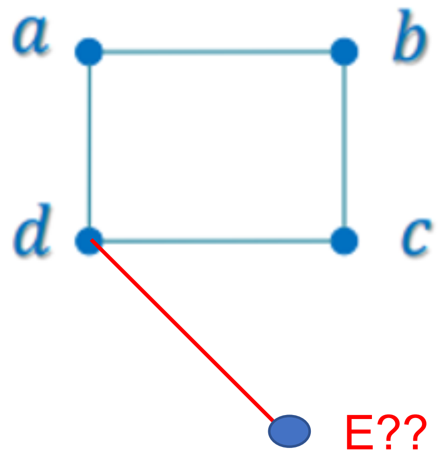


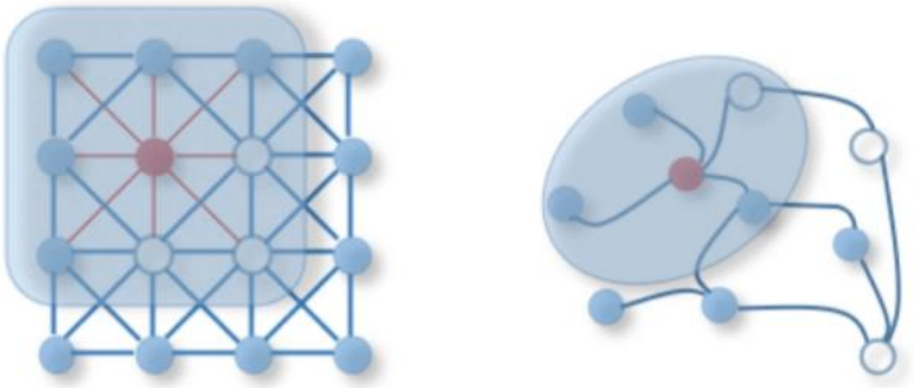
# Backgrounds

**Problem of GCN:**

1. For many applications, test graph may be constantly expanding with new nodes
2. Very Large Computational Cost

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes a weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

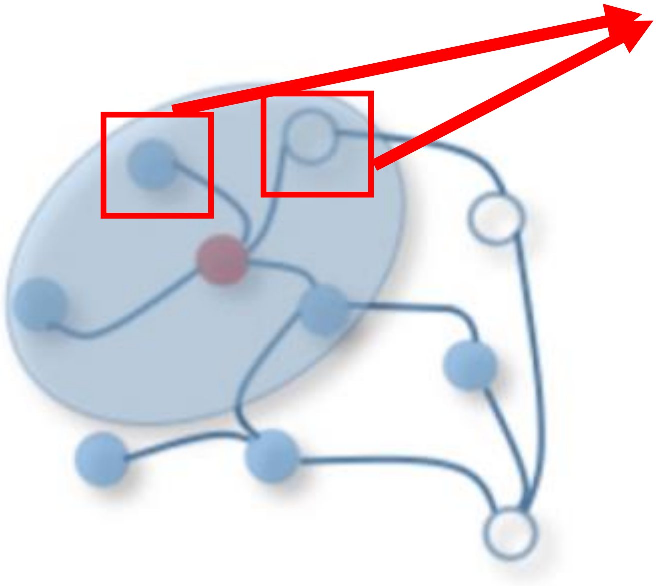
(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of graph convolution operation takes the average value of node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Fig. 1: 2D Convolution vs. Graph Convolution.

# Backgrounds

**Problem of Batch sampling in GCN:**

3. Nodes has high dependency ➔ SGD assumes that all samples are iid



Sampled nodes has dependency

# Method

**Training and Inference through Sampling**

$$\tilde{H}^{(l+1)} = \hat{A}H^{(l)}W^{(l)}, \quad H^{(l+1)} = \sigma(\tilde{H}^{(l+1)}), \quad l = 0, \ldots, M-1,$$

Let function h, s.t

$$\tilde{h}^{(l+1)}(v) = \int \hat{A}(v,u)h^{(l)}(u)W^{(l)}\, dP(u), \quad h^{(l+1)}(v) = \sigma(\tilde{h}^{(l+1)}(v)), \quad l = 0, \ldots, M-1,$$

By Monte Carlo Sampling

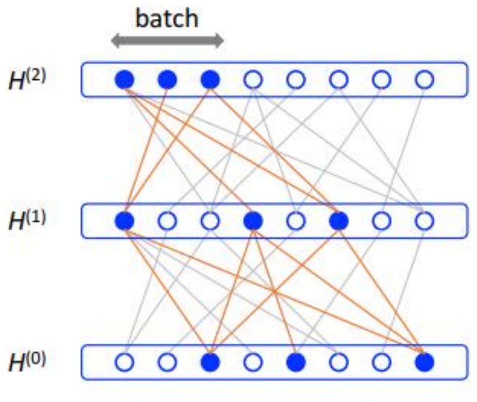
$$\tilde{h}^{(l+1)}_{t_{l+1}}(v) := \frac{1}{t_l}\sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)})h^{(l)}_{t_l}(u_j^{(l)})W^{(l)}, \quad h^{(l+1)}_{t_{l+1}}(v) := \sigma(\tilde{h}^{(l+1)}_{t_{l+1}}(v)), \quad u \sim P$$

Vertex Sampling

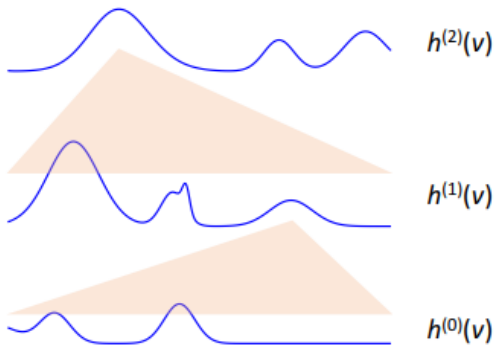
$$H^{(l+1)}(v,:) = \sigma\left(\frac{n}{t_l}\sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)})H^{(l)}(u_j^{(l)},:)W^{(l)}\right), \quad l = 0, \ldots, M-1.$$

Batch Sampling

$$L_{t_0,t_1,\ldots,t_M} := \frac{1}{t_M}\sum_{i=1}^{t_M} g(h^{(M)}_{t_M}(u_i^{(M)}))$$



Graph convolution view



Integral transform view

They assumes that all embedded nodes follows iid ➔ This assumption is too strong …

# Method

*** Adaptive Sampling Toward Fast Graph Representation Learning (NIPS 2018)**

Vertex Sampling

$$H^{(l+1)}(v,:) = \sigma\left(\frac{n}{t_l}\sum_{j=1}^{t_l}\hat{A}(v,u_j^{(l)})H^{(l)}(u_j^{(l)},:)W^{(l)}\right), \quad l = 0,\ldots,M-1.$$

$$pf)\ \sigma\left(\Sigma_j \mathrm{A}(\mathrm{v_i},\mathrm{u_j})\mathrm{h}(\mathrm{u_j})\mathrm{W}\right) = \sigma\left(\frac{n}{1}*\frac{1}{n}\ldots\right) = \sigma\left(n * E_{p(u_j|v_i)}\left(h(u_j)W\right)\right)$$

$$s.t\ p(u_j|v_i) = \frac{A(v_i,u_j)}{n}$$

$$\Rightarrow H = \sigma\left(n * \frac{1}{t_l}\Sigma\ldots\right)$$

# Method

**Training and Inference through Sampling**

**Algorithm 1** FastGCN batched training (one epoch)

1: **for** each batch **do**
2: For each layer $l$, sample uniformly $t_l$ vertices $u_1^{(l)}, \ldots, u_{t_l}^{(l)}$
3: **for** each layer $l$ **do** ▷ Compute batch gradient $\nabla L_{\text{batch}}$
4: If $v$ is sampled in the next layer,

$$\nabla \tilde{H}^{(l+1)}(v,:) \leftarrow \frac{n}{t_l} \sum_{j=1}^{t_l} \hat{A}(v, u_j^{(l)}) \nabla \left\{ H^{(l)}(u_j^{(l)},:) W^{(l)} \right\}$$

5: **end for**
6: $W \leftarrow W - \eta \nabla L_{\text{batch}}$ ▷ SGD step
7: **end for**

# Method

**Variance Reduction by importance sampling**

| | Function | Samples | Num. samples |
|---|---|---|---|
| Layer $l+1$; random variable $v$ | $\tilde{h}^{(l+1)}_{t_{l+1}}(v) \to y(v)$ | $u^{(l+1)}_i \to v_i$ | $t_{l+1} \to s$ |
| Layer $l$; random variable $u$ | $h^{(l)}_{t_l}(u)W^{(l)} \to x(u)$ | $u^{(l)}_j \to u_j$ | $t_l \to t$ |

Under the joint distribution of $v$ and $u$, the aforementioned sample average is

$$G := \frac{1}{s}\sum_{i=1}^{s} y(v_i) = \frac{1}{s}\sum_{i=1}^{s}\left(\frac{1}{t}\sum_{j=1}^{t}\hat{A}(v_i,u_j)x(u_j)\right).$$

$$\mathrm{Var}\{G\} = R + \frac{1}{st}\iint \hat{A}(v,u)^2 x(u)^2\, dP(u)\, dP(v), \tag{6}$$

*where*

$$R = \frac{1}{s}\left(1-\frac{1}{t}\right)\int e(v)^2\, dP(v) - \frac{1}{s}\left(\int e(v)\, dP(v)\right)^2 \quad \textit{and} \quad e(v) = \int \hat{A}(v,u)x(u)\, dP(u).$$

# Method

**Variance Reduction by importance sampling**

Importance Sampling : $E_p\big(f(x)\big) = E_q(\frac{p}{q} f(x))$

**Algorithm 2** FastGCN batched training (one epoch), improved version

For each vertex $u$, compute sampling probability $q(u) \propto \|\hat{A}(:, u)\|^2$
**for** each batch **do**
  For each layer $l$, sample $t_l$ vertices $u_1^{(l)}, \ldots, u_{t_l}^{(l)}$ according to distribution $q$
  **for** each layer $l$ **do** ▷ Compute batch gradient $\nabla L_{\text{batch}}$
    If $v$ is sampled in the next layer,

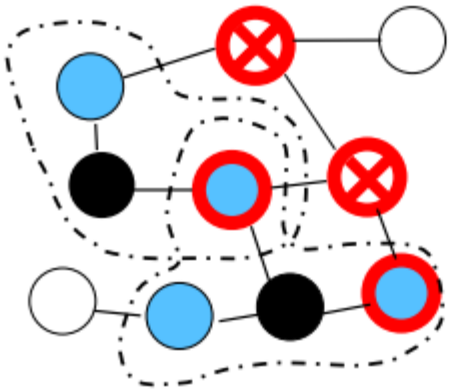
$$\nabla \tilde{H}^{(l+1)}(v, :) \leftarrow \frac{1}{t_l} \sum_{j=1}^{t_l} \frac{\hat{A}(v, u_j^{(l)})}{q(u_j^{(l)})} \nabla \left\{ H^{(l)}(u_j^{(l)}, :) W^{(l)} \right\}$$

  **end for**
  $W \leftarrow W - \eta \nabla L_{\text{batch}}$ ▷ SGD step
**end for**

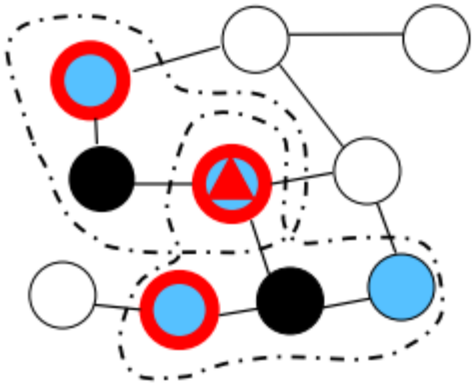
# Discussion

**Problem of FastGCN**

1.



**Layer-wise Importance Sampling (FastGCN)**



**Node-wise Neighbor Sampling (GraphSAGE)**

2. Batch sampling has less theorical background

# Experiments

1Classifying research topics using Cora citation data set

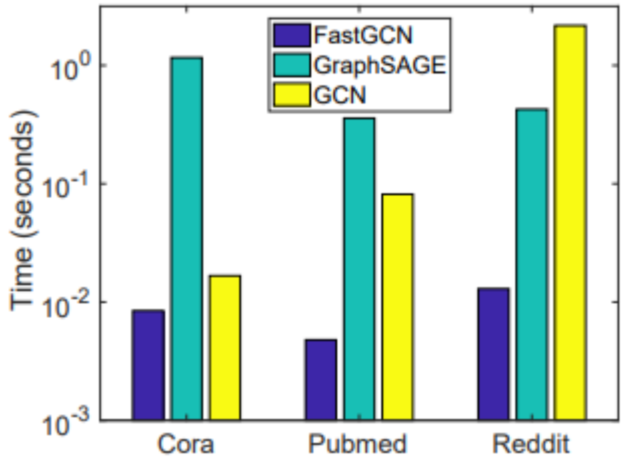
2 Categorizing academic papers with the Pubmed database

3 Predicting the community structure of a social network modeled with Reddit posts

Table 1: Dataset Statistics

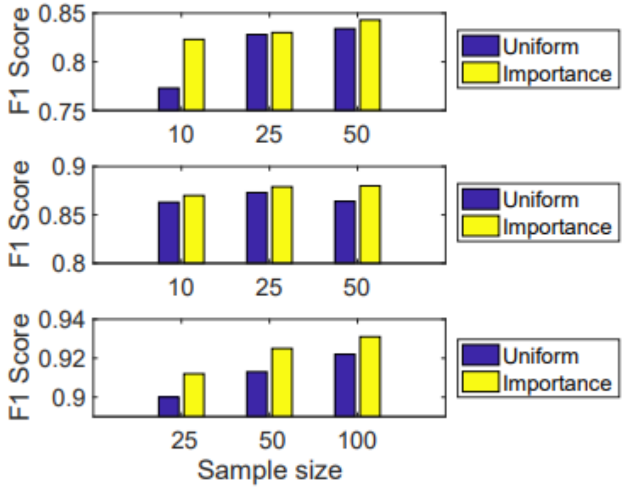
| Dataset | Nodes | Edges | Classes | Features | Training/Validation/Test |
|---|---|---|---|---|---|
| Cora | 2,708 | 5,429 | 7 | 1,433 | 1,208/500/1,000 |
| Pubmed | 19,717 | 44,338 | 3 | 500 | 18,217/500/1,000 |
| Reddit | 232,965 | 11,606,919 | 41 | 602 | 152,410/23,699/55,334 |

# Experiments




| Micro F1 Score | Cora | Pubmed | Reddit |
|---|---|---|---|
| FastGCN | 0.850 | 0.880 | 0.937 |
| GraphSAGE-GCN | 0.829 | 0.849 | 0.923 |
| GraphSAGE-mean | 0.822 | 0.888 | 0.946 |
| GCN (batched) | 0.851 | 0.867 | 0.930 |
| GCN (original) | 0.865 | 0.875 | NA |

| | Sampling | | Precompute | |
|---|---|---|---|---|
| $t_1$ | Time | F1 | Time | F1 |
| 5 | 0.737 | 0.859 | 0.139 | 0.849 |
| 10 | 0.755 | 0.863 | 0.141 | 0.870 |
| 25 | 0.760 | 0.873 | 0.144 | 0.879 |
| 50 | 0.774 | 0.864 | 0.142 | 0.880 |



# Thank you