HyperGCN:
A New Method of Training Graph Convolutional Networks on Hypergraphs

N Yadati, et al
NIPS 2019

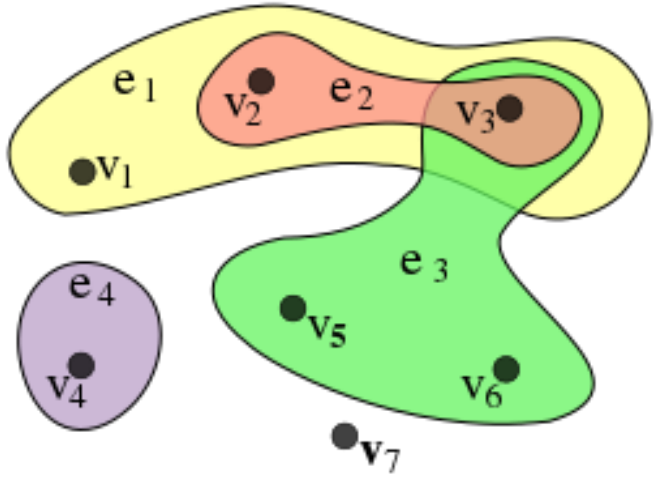
Jun Ha Chun

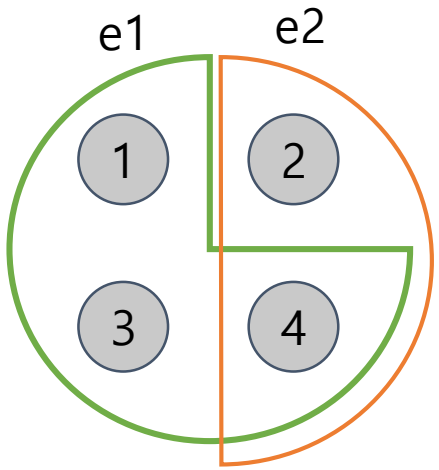
Seoul National University

Contents

Introduction - Hypergraph

- Generalization of a graph
  - Hyperedge can join any number of nodes
  - Examples
    - Co-citation
    - Co-authorship
    - Chemical reaction
    - 3D point cloud
  - Tasks (like graphs)
    - SSL
    - Combinatorial optimization





$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Backgrounds - Remind

- GCN
  - $\mathcal{G} = (V, E),\ |V| = n,\ |E| = m,\ V_L \subset V$
  - $\bar{A} = \widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}, \tilde{A} = A + I, \widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
  - GCN: $\sigma(\bar{A}X\Theta),\ X \in \mathbb{R}^{n\times p}, \Theta \in \mathbb{R}^{p\times r}$

$$Z = f_{GCN}(X, A) = \mathrm{softmax}\left(\bar{A}\ \mathrm{ReLU}\left(\bar{A}X\Theta^{(1)}\right)\Theta^{(2)}\right)$$

- SSL
  - Cross-entropy $\quad \mathcal{L} = -\sum_{i\in\mathcal{V}_L}\sum_{j=1}^{q} Y_{ij}\ln Z_{ij}.$

Backgrounds - Hypergraph

- Definition
  - $\mathcal{H} = (V, E),\ |V| = n,\ |E| = m,\ V_L \subset V$
  - $x_v \in \mathbb{R}^p, X \in \mathbb{R}^{n \times p}, H = 1^{n \times m}$
- Normalized Hypergraph Cut

$$\underset{\emptyset \neq S \subset V}{\operatorname{argmin}} c(S) := \operatorname{vol} \partial S \left(\frac{1}{\operatorname{vol} S} + \frac{1}{\operatorname{vol} S^c} \right)$$

- Relaxation with constraints

$$\underset{f \in \mathbb{R}^{|V|}}{\operatorname{argmin}} \frac{1}{2} \sum_{e \in E} \sum_{\{u,v\} \subseteq e} \frac{w(e)}{\delta(e)} \left(\frac{f(u)}{\sqrt{d(u)}} - \frac{f(v)}{\sqrt{d(v)}} \right)^2 = f^T \Delta f.$$

$$\text{subject to} \sum_{v \in V} f^2(v) = 1,\ \sum_{v \in V} f(v) \sqrt{d(v)} = 0.$$

$$\Theta = D_v^{-1/2} H W D_e^{-1} H^T D_v^{-1/2} \text{ and } \Delta = I - \Theta$$

Backgrounds - Hypergraph

- Hypergraph convolution(HGNN)
  - $\Delta = I - D_v^{-\frac{1}{2}} H W D_e^{-1} H^T D_v^{-\frac{1}{2}}$

$$\mathbf{g} \star \mathbf{x} = \mathbf{\Phi}((\mathbf{\Phi}^\top \mathbf{g}) \odot (\mathbf{\Phi}^\top \mathbf{x})) = \mathbf{\Phi} g(\mathbf{\Lambda}) \mathbf{\Phi}^\top \mathbf{x} \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{\Delta}}) \mathbf{x},$$

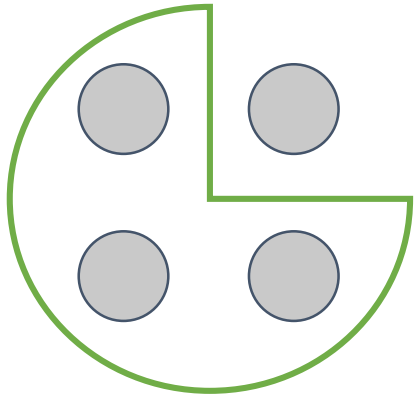
$$\approx \theta_0 \mathbf{x} - \theta_1 \mathbf{D}^{-1/2} \mathbf{H} \mathbf{W} \mathbf{D}_e^{-1} \mathbf{H}^\top \mathbf{D}_v^{-1/2} \mathbf{x},$$

$$\approx \frac{1}{2} \theta \mathbf{D}_v^{-1/2} \mathbf{H} (\mathbf{W} + \mathbf{I}) \mathbf{D}_e^{-1} \mathbf{H}^\top \mathbf{D}_v^{-1/2} \mathbf{x}$$

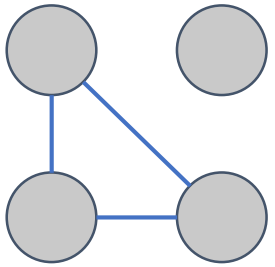
$$\approx \theta \mathbf{D}_v^{-1/2} \mathbf{H} \mathbf{W} \mathbf{D}_e^{-1} \mathbf{H}^\top \mathbf{D}_v^{-1/2} \mathbf{x},$$

$$\mathbf{Y} = \mathbf{D}_v^{-1/2} \mathbf{H} \mathbf{W} \mathbf{D}_e^{-1} \mathbf{H}^\top \mathbf{D}_v^{-1/2} \mathbf{X} \mathbf{\Theta},$$

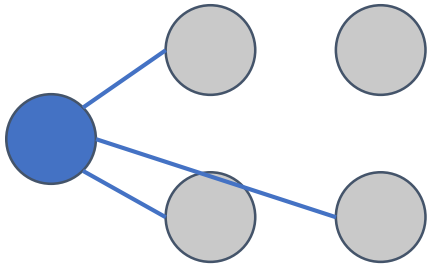
Backgrounds - Hypergraph



- Hypergraph approximation
  - Hypergraph to graph
  - Clique expansion
    - Hyperedge to Clique
      - $O(s^2)$ edges

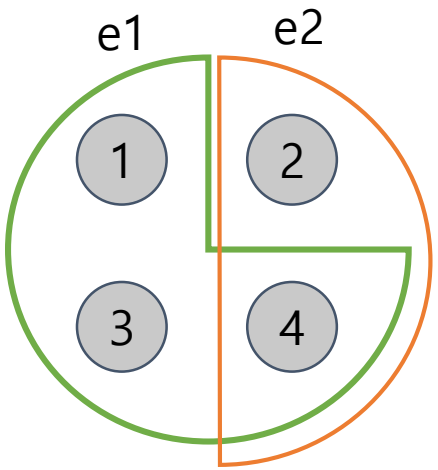


  - Star expansion
    - Hyperedge to
      - a new vertex
      - edges connecting the new vertex to each vertex in the hyperedge
      - $O(s)$ edges



Backgrounds - Hypergraph

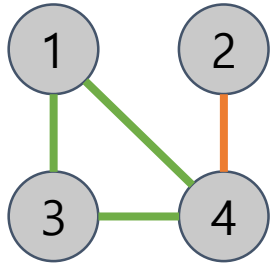
- Hypergraph approximation
  - $\Delta = I - D_v^{-\frac{1}{2}} HWD_e^{-1}H^T D_v^{-\frac{1}{2}}$
  - $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$



$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$HH^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

- Hypergraph Laplacian implies clique expansion
  - Inefficient when hyperedges are large



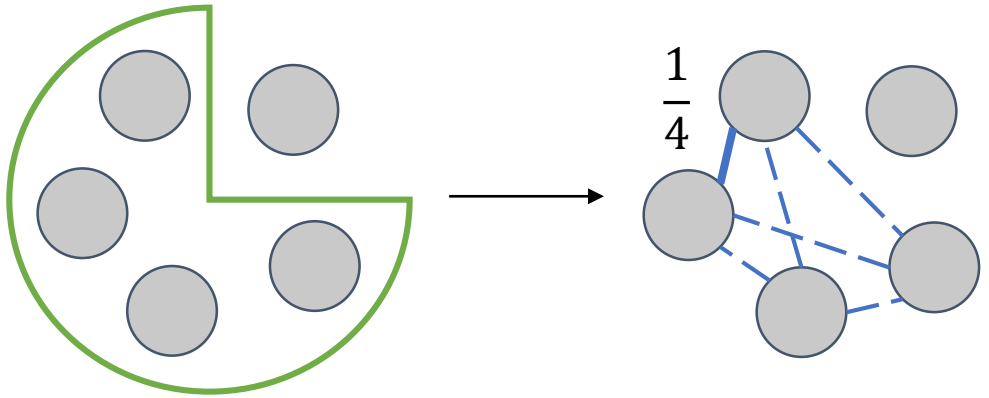
HyperGCN

- Let's reduce order of edges added for each hyperedge
  - Explicit hyperedge expansion
  - Select all edge = Clique
  - Select some edges?
  - Select 1 edge?

- What do we learn?
  - SSL node classification setting: nodes in hyperedge should be similar
  - Smoothness regularizer: $\sum_{e \in E} \max_{i,j \in e} \left| h_i - h_j \right|^2$
  - Let's select with $\underset{i,j \in e}{argmax} \left| h_i - h_j \right|^2$
  - Edges with large difference should be "learned" more

HyperGCN

- **1-HyperGCN** (select 1 edge)
  - $G_S = \{V, E_S\}, E_S = \left\{ \underset{i,j \in e}{argmax} \left| h_i - h_j \right|^2 : e \in E \right\} + self\ edges$
  - $w(\{i_e, j_e\}) = \frac{1}{|e|}$, $A_S =$ $weighted\ adjacency\ matrix$
  - GCN step: $\sigma(\overline{A_S} X \Theta)$



HyperGCN

- 1-HyperGCN

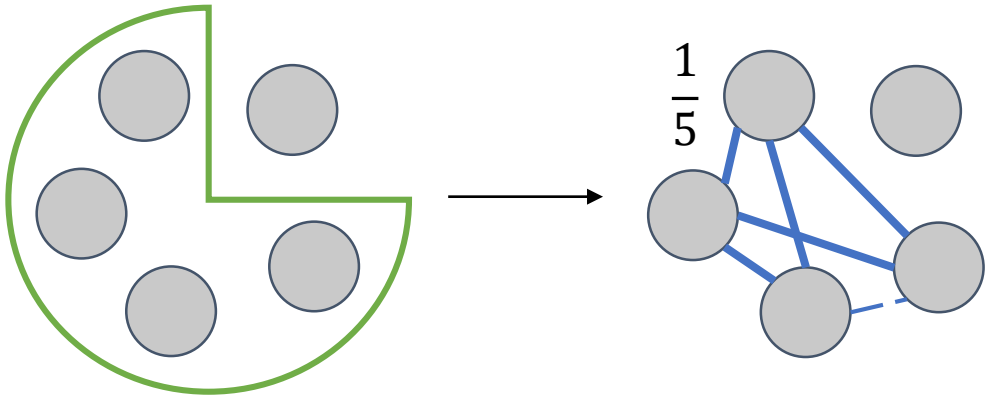
Algorithm 3 Algorithm for 1-HyperGCN

Input: An attributed hypergraph $\mathcal{H} = (V, E, X)$, with attributes X, a set of labelled vertices $\mathcal{V}_L$
Output All hypernodes in $V - \mathcal{V}_L$ labelled

for each epoch τ of training **do**
  for layer $l = 1, 2$ of the network **do**
    set $A^{(l)}_{vv} = 1$ for all hypernodes $v \in V$
    let $\Theta = \Theta^{\tau}$ be the parameters for the current epoch
    for $e \in E$ **do**
      $H \leftarrow$ hidden representation matrix of layer $l - 1$
      $i_e, j_e := \text{argmax}_{i,j \in e} ||H_i(\Theta^{(l)}) - H_j(\Theta^{(l)})||_2$
      $A^{(l)}_{i_e, j_e} = A^{(l)}_{j_e, i_e} = \frac{1}{|e|}$
    end for
  end for
  $Z = \text{softmax}\left(\bar{A}^{(2)} \ \text{ReLU}\left(\bar{A}^{(1)} X \Theta^{(1)}\right)\Theta^{(2)}\right)$
  update parameters Θ^{τ} to minimise cross entropy loss on the set of labelled hypernodes $\mathcal{V}_L$
end for
label the hypernodes in $V - \mathcal{V}_L$ using Z

HyperGCN

- One is not enough: ignores nodes in $K_e = \{k \in e : k \neq i_e, k \neq j_e\}$
- Enhance by using them as mediators
- **HyperGCN** (Select more)
  - $G_S = \{V, E_S\}, E_S = \left\{ \underset{i,j \in e}{argmax} \left| h_i - h_j \right|^2 : e \in E \right\}$
  - $+\{(k,l): l \in \{i_e, j_e\}, k \in K_e,\ e \in E\} + self\ edges$
  - $w(\{i_e, j_e\}) = \frac{1}{2|e|-3}$, $A_S =$ $weighted\ adjacency\ matrix$
  - GCN step: $\sigma(\overline{A_S} X \Theta)$



Algorithm 1 Algorithm for HyperGCN

Input: An attributed hypergraph $\mathcal{H} = (V, E, X)$, with attributes X, a set of labelled vertices $\mathcal{V}_L$
Output All hypernodes in $V - \mathcal{V}_L$ labelled

for each epoch τ of training **do**
  for layer $l = 1, 2$ of the network **do**
    set $A^{(l)}_{vv} = 1$ For all hypernodes $v \in V$
    let $\Theta = \Theta^{\tau}$ be the parameters For the current epoch
    for $e \in E$ **do**
      $H \leftarrow$ hidden representation matrix of layer $l - 1$
      $i_e, j_e := \text{argmax}_{i,j \in e} ||H_i(\Theta^{(l)}) - H_j(\Theta^{(l)})||_2$
      $A^{(l)}_{i_e,j_e} = A^{(l)}_{j_e,i_e} = \frac{1}{2|e|-3}$
      $K_e := \{k \in e : \ k \neq i_e, k \neq j_e\}$
      for $k \in K_e$ **do**
        $A^{(l)}_{i_e,k} = A^{(l)}_{k,i_e} = \frac{1}{2|e|-3}$
        $A^{(l)}_{j_e,k} = A^{(l)}_{k,j_e} = \frac{1}{2|e|-3}$
      end for
    end for
  end for
  $Z = \text{softmax}\left(\bar{A}^{(2)} \ \text{ReLU}\left(\bar{A}^{(1)} X \Theta^{(1)}\right)\Theta^{(2)}\right)$
  update parameters Θ^{τ} to minimise cross entropy loss on the set of labelled hypernodes $\mathcal{V}_L$
end for
label the hypernodes in $V - \mathcal{V}_L$ using Z

HyperGCN

- G_S is recomputed every epoch
  - To learn different structures as parameter changes

- **FastHyperGCN**
  - Use initial features X to construct fixed Laplacian

Algorithm 2 Algorithm for FastHyperGCN

Input: An attributed hypergraph $\mathcal{H} = (V, E, X)$, with attributes X, a set of labelled vertices $\mathcal{V}_L$
Output All hypernodes in $V - \mathcal{V}_L$ labelled

set $A_{vv} = 1$ for all hypernodes $v \in V$
$i_e, j_e := \text{argmax}_{i,j \in e} ||X_i - X_j||_2$
for $e \in E$ **do**
  $A_{i_e,j_e} = A_{j_e,i_e} = \frac{1}{2|e|-3}$
  $K_e := \{k \in e : \ k \neq i_e, k \neq j_e\}$
  for $k \in K_e$ **do**
    $A_{i_e,k} = A_{k,i_e} = \frac{1}{2|e|-3}$
    $A_{j_e,k} = A_{k,j_e} = \frac{1}{2|e|-3}$
  end for
end for
for each epoch τ of training **do**
  let $\Theta = \Theta^\tau$ be the parameters for the current epoch
  $$Z = \text{softmax}\left(\bar{A}\ \text{ReLU}\left(\bar{A}X\Theta^{(1)}\right)\Theta^{(2)}\right)$$
  update parameters Θ^τ to minimise cross entropy loss on the set of labelled hypernodes $\mathcal{V}_L$
end for
label the hypernodes in $V - \mathcal{V}_L$ using Z

Experiments

- Dataset: Co-citation/authorship hypergraph

	DBLP (co-authorship)	**Pubmed** (co-citation)	**Cora** (co-authorship)	**Cora** (co-citation)	**Citeseer** (co-citation)
# hypernodes, $\|V\|$	43413	19717	2708	2708	3312
# hyperedges, $\|E\|$	22535	7963	1072	1579	1079
avg.hyperedge size	$\mathbf{4.7 \pm 6.1}$	$\mathbf{4.3 \pm 5.7}$	$\mathbf{4.2 \pm 4.1}$	$\mathbf{3.0 \pm 1.1}$	$\mathbf{3.2 \pm 2.0}$
# features, d	1425	500	1433	1433	3703
# classes, q	6	3	7	7	6
label rate, $\|V_L\|/\|V\|$	0.040	0.008	0.052	0.052	0.042

- SSL: Hypernode classification task
- Combinatorial Optimization: K-subhypergraph problem
  - Find subset W $\subseteq V$ of k hypernodes s.t. maximize # of hyperedges in W (NPhard)

Experiments

- SSL: Hypernode classification task

Data	Method	DBLP co-authorship	Pubmed co-citation	Cora co-authorship	Cora co-citation	Citeseer co-citation
$\mathcal{H}$	CI	54.81 ± 0.9	52.96 ± 0.8	55.45 ± 0.6	64.40 ± 0.8	70.37 ± 0.3
$\mathbf{X}$	MLP	37.77 ± 2.0	30.70 ± 1.6	41.25 ± 1.9	42.14 ± 1.8	41.12 ± 1.7
$\mathcal{H}, \mathbf{X}$	MLP + HLR	30.42 ± 2.1	30.18 ± 1.5	34.87 ± 1.8	36.98 ± 1.8	37.75 ± 1.6
$\mathcal{H}, \mathbf{X}$	HGNN	25.65 ± 2.1	29.41 ± 1.5	31.90 ± 1.9	$\mathbf{32.41 \pm 1.8}$	$\mathbf{37.40 \pm 1.6}$
$\mathcal{H}, \mathbf{X}$	1-HyperGCN	33.87 ± 2.4	30.08 ± 1.5	36.22 ± 2.2	34.45 ± 2.1	38.87 ± 1.9
$\mathcal{H}, \mathbf{X}$	FastHyperGCN	27.34 ± 2.1	29.48 ± 1.6	32.54 ± 1.8	$\mathbf{32.43 \pm 1.8}$	$\mathbf{37.42 \pm 1.7}$
$\mathcal{H}, \mathbf{X}$	HyperGCN	$\mathbf{24.09 \pm 2.0}$	$\mathbf{25.56 \pm 1.6}$	$\mathbf{30.08 \pm 1.8}$	$\mathbf{32.37 \pm 1.7}$	$\mathbf{37.35 \pm 1.6}$

  - Clique expansion = expansion with mediators when $|e| = 2, 3$

Experiments

- SSL: Hypernode classification task
  - Robust to noisiness ($low\ \eta = noisy$)

Method	$\eta = 0.75$	$\eta = 0.70$	$\eta = 0.65$	$\eta = 0.60$	$\eta = 0.55$	$\eta = 0.50$	sDBLP
HGNN	**15.92 ± 2.4**	**24.89 ± 2.2**	**31.32 ± 1.9**	39.13 ± 1.78	42.23 ± 1.9	44.25 ± 1.8	45.27 ± 2.4
FastHyperGCN	28.86 ± 2.6	31.56 ± 2.7	33.78 ± 2.1	33.89 ± 2.0	**34.56 ± 2.2**	**35.65 ± 2.1**	41.79 ± 2.8
HyperGCN	22.44 ± 2.0	29.33 ± 2.2	33.41 ± 1.9	**33.67 ± 1.9**	35.05 ± 2.0	37.89 ± 1.9	**41.64 ± 2.6**

  - Training time

Model↓ Metric →	Training time	Density	DBLP	Pubmed
HGNN	170s	337	0.115s	0.019s
FastHyperGCN	**143s**	**352**	**0.035s**	**0.016s**

Table 1: average training time of an epoch (lower is better)

Experiments

- Combinatorial Optimization: K-subhypergraph problem
  - Maxmizing “density”
  - Greedy heuristic
    - MaxDegree: select k nodes with largest degree
    - RemoveMinDegree: remove all hyperedge including smallest degree node (repeat n-k times)

Dataset→ Approach↓	**Synthetic** test set	**DBLP** co-authorship	**Pubmed** co-citation	**Cora** co-authorship	**Cora** co-citation	**Citeseer** co-citation
MaxDegree	174 ± 50	4840	1306	194	544	507
RemoveMinDegree	147 ± 48	**7714**	**7963**	450	1369	843
MLP	174 ± 56	5580	1206	238	550	534
MLP + HLR	231 ± 46	5821	3462	297	952	764
HGNN	337 ± 49	6274	7865	437	**1408**	**969**
1-HyperGCN	207 ± 52	5624	1761	251	563	509
FastHyperGCN	352 ± 45	7342	7893	452	**1419**	**969**
HyperGCN	$\mathbf{359 \pm 49}$	**7720**	**7928**	**504**	**1431**	**971**
# hyperedges, $\lvert E \rvert$	500	22535	7963	1072	1579	1079

Thank you

Experiments - reproducing

- Settings
  - SSL Node classification
    - Dataset: DBLP co-authorship

      CORA co-citation

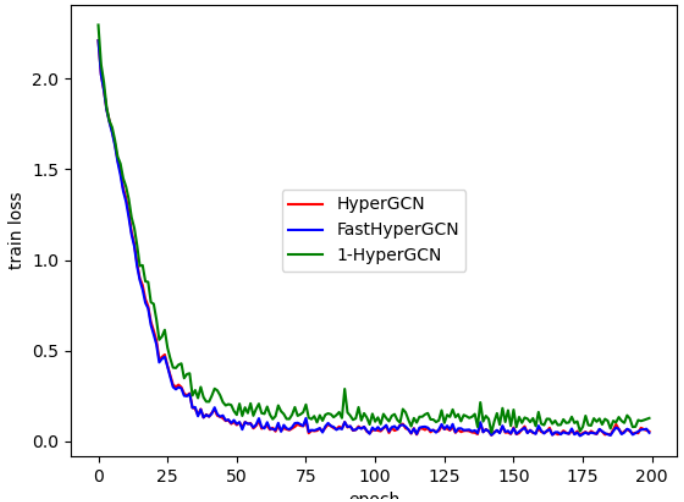
    - Parameters
      - lr=0.01, weight_decay=5e-4, epochs=200
      - hidden=16, dropout=0.5

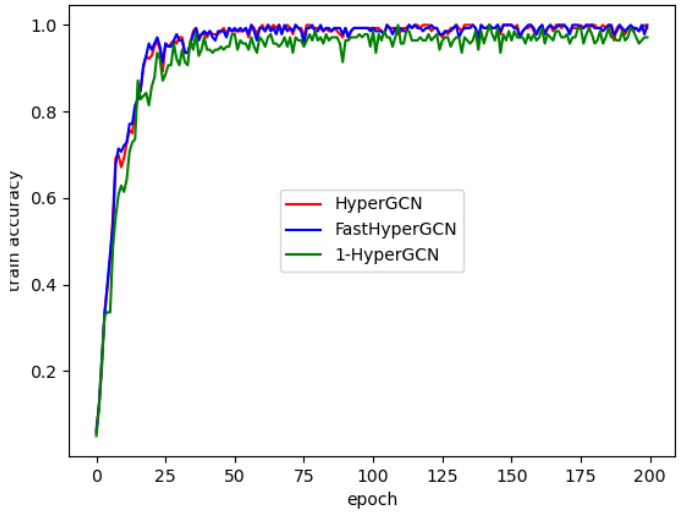
```
[DBLP] # of nodes: 41302
[DBLP] Dim of feature: 1425
[DBLP] # of hyperedges: 22363
[DBLP] # of classes: 41302
[DBLP] Labels: [0 1 2 3 4 5]
[DBLP] # of train: 1740
[DBLP] # of val: 39562
[MODEL] in, hid, out: [1425, 16, 6]
[OPTIM] lr, decay: 0.01, 0.0005
```

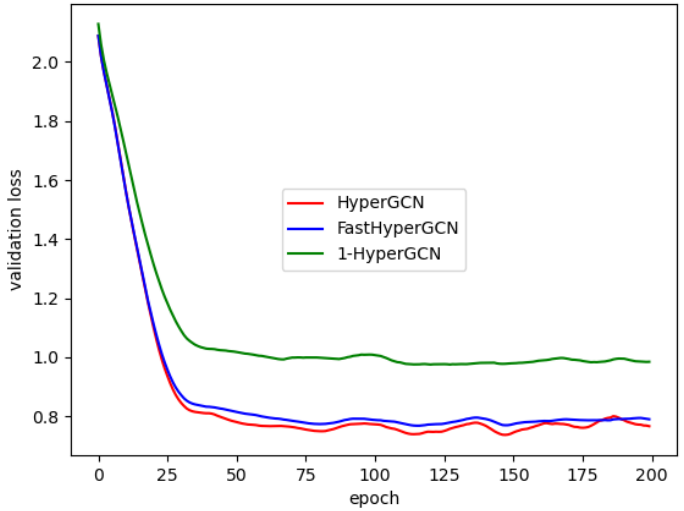
```
[CORA] # of nodes: 2708
[CORA] Dim of feature: 1433
[CORA] # of hyperedges: 1564
[CORA] # of classes: 7
[CORA] Labels: [0 1 2 3 4 5 6]
[CORA] # of train: 140
[CORA] # of val: 500
[CORA] # of test: 2068
[MODEL] in, hid, out: [1433, 16, 7]
[OPTIM] lr, decay: 0.01, 0.0005
```

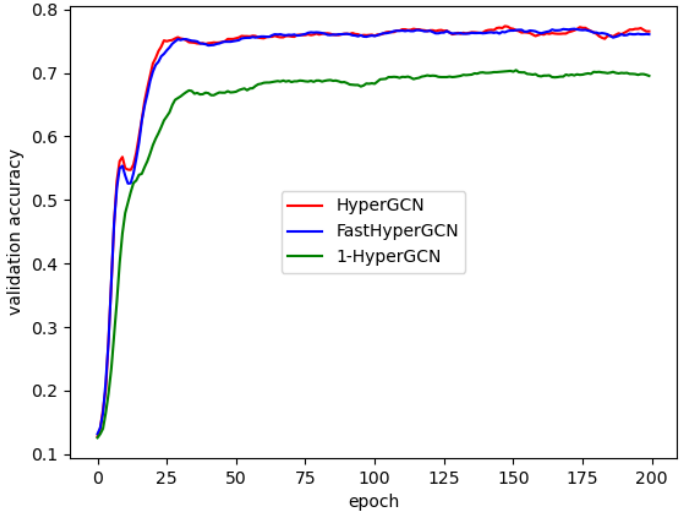
Experiments - reproducing

- Cora results



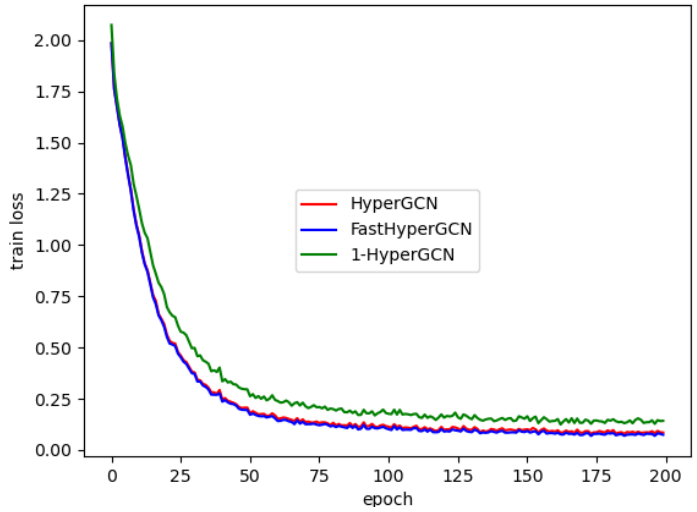


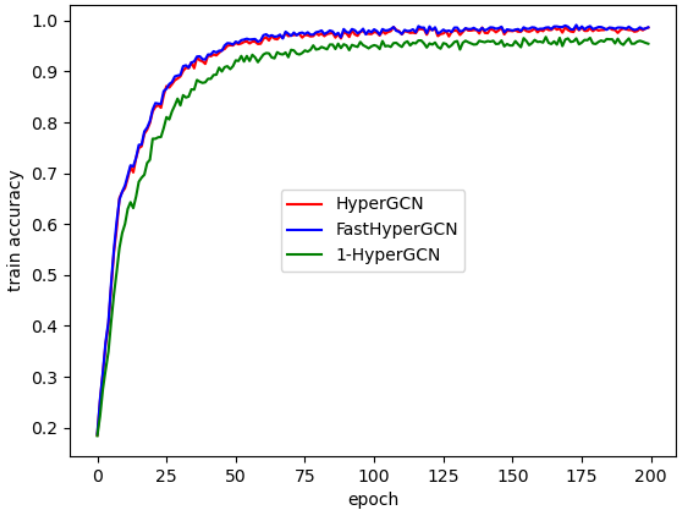


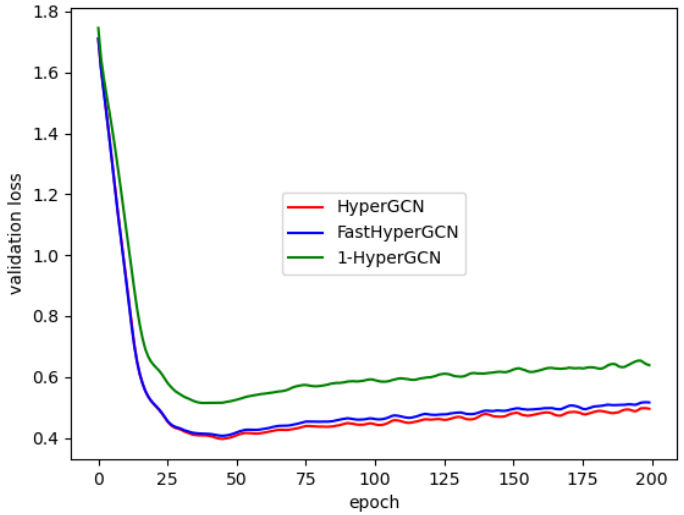


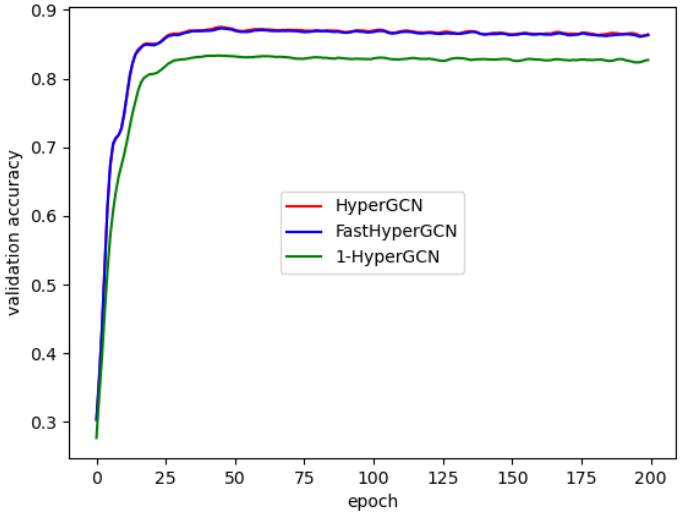
Experiments - reproducing

- DBLP results









Experiments - reproducing

- Results in the paper is not reliable
  - Train-test split experiments
    - While Train-val-test should be done
  - Numbers in the table: 1 – accuracy
    - Too high error, experiment settings are not clear

Data	Method	DBLP co-authorship	Cora co-authorship
$\mathcal{H}, \mathbf{X}$	1-HyperGCN	33.87 ± 2.4	36.22 ± 2.2
$\mathcal{H}, \mathbf{X}$	FastHyperGCN	27.34 ± 2.1	32.54 ± 1.8
$\mathcal{H}, \mathbf{X}$	HyperGCN	$\mathbf{24.09 \pm 2.0}$	$\mathbf{30.08 \pm 1.8}$

- It reached certain performance
  - Near 77.3%(cora), 87.5%(dblp)
  - But worse than reported performances from other works
    - GCN(81.5%, cora), HGNN(81.6%, cora)
  - There might be bugs in my code
    - but also couldn't reach the performance from official public code

Repository

- Link: https://github.com/nikriz/HyperGCN
- How to run (README.md)

```
conda env create –f env.yaml
conda activate hypergcn
python train.py    --dataset      <“cora” or “dblp”> to choose the dataset
                   --split_id     <1~10> to choose the split, for dblp
                   --one/fast     to choose the 1-HyperGCN/FastHyperGCN
                   --epochs       <value> to choose training epochs
                   --lr           <value> to choose learning rate
                   --weight_decay <value> to choose weight decay
                   --hidden       <value> to choose hidden layer size
                   --dropout      <value> to choose dropout rate
```

Implementation

- Code structure
  - Model.py: basic GCN model

    ref) https://github.com/tkipf/pygcn/blob/master/pygcn/models.py
  - Train.py: training code
  - Util.py: graph utility functions and graph update

    ref) https://github.com/malllabiisc/HyperGCN/blob/master/model/utils.py
  - Data/: dataset and load scripts

Implementation

- Model.py
  - Graph Convolution
    - Same as reference
      - reset_parameters
        - Gaussian initialization
      - forward
        - (output) = A(input)Θ+b

```
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """
    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features,
        out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' (' \
               + str(self.in_features) + ' -> ' \
               + str(self.out_features) + ')'
```

Implementation

- Model.py
  - 2-layer GCN module
    - Same as reference
      - Softmax(gc2(Dropout(Relu(gc1))))
    - Little modifications
      - To use intermediate features in graph structure update, return layers argument is added to retrieve features from wanted layer.

```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()
        print("[MODEL] in, hid, out:", [nfeat, nhid, nclass])
        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)
        self.layers = nn.ModuleList([self.gc1, self.gc2])
        self.dropout = dropout

    def forward(self, x, adjs, return_layer=2):
        if return_layer is 0: return x
        x = F.relu(self.layers[0](x, adjs[0]))
        if return_layer is 1: return x
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.layers[1](x, adjs[1])
        return F.log_softmax(x, dim=1)
```

Implementation

- Util.py
  - Utility functions
    - Same as reference
      - Symnormalize: $D^{-\frac{1}{2}}MD^{-\frac{1}{2}}$
    - Referenced, but not copied
      - Format modifiers

```python
# Symnormalize matrix
def symnormalise(M):
    d = np.array(M.sum(1))
    dhi = np.power(d, -1/2).flatten()
    dhi[np.isinf(dhi)] = 0.
    DHI = sp.diags(dhi)
    return (DHI.dot(M)).dot(DHI)

# Dictionary of edges to normalized adjacency matrix
def dictionary_to_sparse_mx(dictionary, n):
    edges = [list(k) for k in dictionary.keys()]
    weights = [dictionary[k] for k in dictionary.keys()]
    edges = np.array(edges)
    weights = np.array(weights)
    adj = sp.coo_matrix((weights, (edges[:, 0], edges[:, 1])), shape=(n,
    n), dtype=np.float32)
    adj = adj + sp.eye(n)
    adj = symnormalise(sp.csr_matrix(adj, dtype=np.float32))
    return adj

# Sparse matrix to sparse tensor
def sparse_mx_to_torch_sparse_tensor(sparse_mx):
    sparse_mx = sparse_mx.tocoo().astype(np.float32)
    indices = torch.from_numpy(
        np.vstack((sparse_mx.row, sparse_mx.col)).astype(np.int64))
    values = torch.from_numpy(sparse_mx.data)
    shape = torch.Size(sparse_mx.shape)
    return torch.sparse.FloatTensor(indices, values, shape)
```

Implementation

- Util.py
  - Utility functions
    - Same as reference
      - Evaluation function
    - Not from reference
      - Sparse Identity generator

```python
# Get accuracy of result
def accuracy(output, labels):
    preds = output.max(1)[1].type_as(labels)
    correct = preds.eq(labels).double()
    correct = correct.sum()
    return correct / len(labels)
```

```python
# Get identity matrix
def eye(n, to_tensor=False):
    m = {}
    for i in range(n):
        m[(i, i)] = 0
    if to_tensor:
        m = dictionary_to_sparse_mx(m, n)
        m = sparse_mx_to_torch_sparse_tensor(m)
    return m
```

Implementation

- Util.py
  - Graph Updater
    - Refactored reference
      - Sparse dictionary edge adder
      - Graph updating function

```python
# Link node i and j with weight w
def link(adj, i, j, w):
    if (i, j) not in adj:
        adj[(i, j)] = 0
    adj[(i, j)] += w
    if (j, i) not in adj:
        adj[(j, i)] = 0
    adj[(j, i)] += w

# Update graph structure for hypergcn
def update_graph(hypergraph, features, one=False):
    print("[UPDATE]")
    n = features.shape[0]
    adj = eye(n)
    cnt = 0
    for author in hypergraph:
        hyperedge = np.array(hypergraph[author])

        # Get max feature diff
        with torch.no_grad():
            f = np.array(features[hyperedge])
        dist = cdist(f, f)
        idx = np.unravel_index(np.argmax(dist, axis=None), dist.shape)

        # Connect edges
        weight = 1/(2*len(hyperedge) - 3)
        link(adj, hyperedge[idx[0]], hyperedge[idx[1]], weight)

        # Only for Hypergcn/FastHypergcn
        if not one:
            for i in hyperedge:
                if i not in [hyperedge[idx[0]], hyperedge[idx[1]]]:
                    link(adj, hyperedge[idx[0]], i, weight)
                    link(adj, hyperedge[idx[1]], i, weight)
                    cnt = cnt + 2
        cnt = cnt + 1

    # return graph structure
    adj = dictionary_to_sparse_mx(adj, n)
    adj = sparse_mx_to_torch_sparse_tensor(adj)
    return adj
```

Again Thank you