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

Aman Saxena	2016B2A80931P
Aditya Rustagi	2016B2A30857P
Avishek Roy	2017A3PS1163P

Problem?

- We consider the problem of classifying nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes.
- This problem is framed as graph-based semi-supervised learning.
- The paper by Kipf & Welling utilizes Tensorflow but for our problem, we utilized PyTorch because it is quick and easy to learn. *PyTorch has a simple Python interface and provides a simple yet powerful API.*
- *Why GCN for our problem?*
  GCN's allow algorithms to analyze information in its native form rather than requiring an arbitrary representation of that same information in lower dimension space which destroys relationships and thus information.

Why Graphs?

Graphs?

- In most Machine Learning applications, data can be more naturally represented as graphs, more effectively. Even neural network itself can be viewed as a graphs.

Graph Neural Networks?

- Graph Neural Networks are very flexible conceptually, but the problem is that in graphs there is no well-defined order of nodes.

- The difficulty of regularizing the model is being solved by defining new operators as convolution for graphs. eg. aggregate operator should be permutation invariant.

Theory?

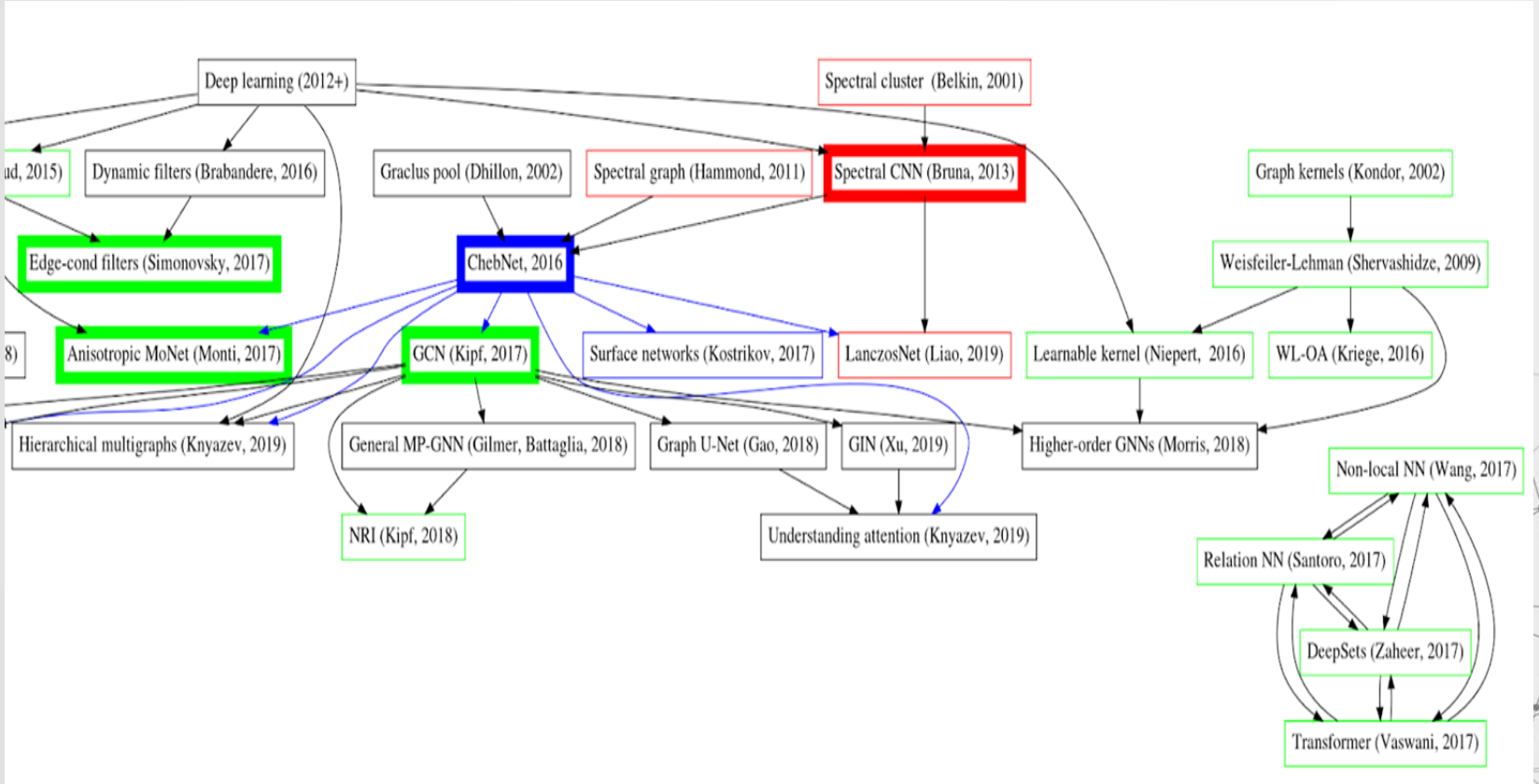
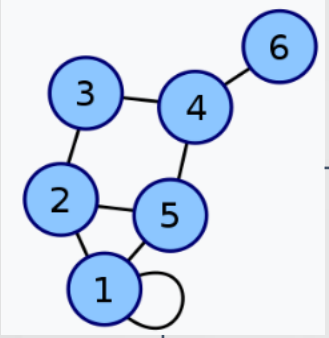


Image source: [1]

Concept?



$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Adjacency Matrix ⇨ Indicates nearby nodes

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

Degree Matrix ⇨ Diagonal matrix indicating total number of nearby nodes.

Theory?

Spectral Graph Convolutions?

- For graphs and graph neural networks (GNNs), “spectral” implies eigen-decomposition of the graph Laplacian “L”, it defines how node features will be updated if we stack several graph neural layers.

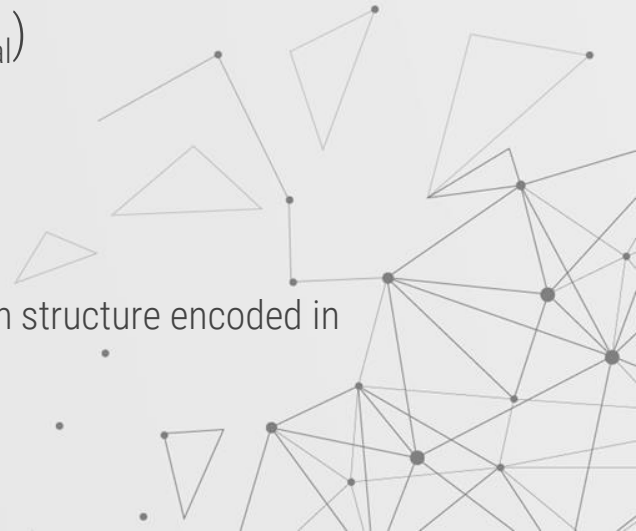
$$L=I_N-D^{-1/2}AD^{-1/2} \qquad L=U\Lambda U^T$$

- In spectral convolutions, $Output_{l+1} = X_{l+1}=U(U^TX_l.U^TW_{spectral})$

- Convolution of a filter on graph is defined as

$$g_\theta * x = Ug_\theta U^T x$$

- g_θ depends on the number of nodes N in a graph and the graph structure encoded in eigenvectors U.



Theory?

Smooth Filters?

- Bruna et al. proposed to smooth filters in the spectral domain.
- The idea is that to represent our filter from formula as a sum of K predefined functions, and instead of learning N values of W, we learn K coefficients.

$$W^{l}_{spectral} \approx \sum_{k=1}^{K} \alpha_k f_k$$

- we want K<<N to reduce the number of trainable parameters from N to K so that our GNN can train on graphs of any size.
- g_θ still depends on graph structure encoded in eigenvectors.

Theory?

Chebyshev?

- Hammond et al. suggested that filters can be well-approximated by a truncated expansion in terms of Chebyshev polynomials. The Chebyshev polynomials are defined in recurrence relation terms with with $T_0(x) = 1$ and $T_1(x) = x$.

$$g_\theta(\Lambda) \approx {}^{K}\sum_{k=0} \theta'_k T_k(\tilde{\Lambda})$$

- Using Chebyshev approximation we get

$$g_\theta * x \approx {}^{K}\sum_{k=0} \theta'_k T_k(\hat{L})x \qquad \text{where} \qquad \hat{L} = (2/\lambda_{max})L - I_N$$

- Slow and overfits data as graph data sets are relatively small.

Theory?

Linear Model(Kipf & Welling, ICLR, 2017**)**

- With K=1 and $\lambda_{max}=2$ we get a linear relation in Laplacian L.

$$g_\theta * x \approx \theta_0' x + \theta_1' (L - I_N) x$$

- With $\theta_0' = -\theta_1'$ and $L = I_N - D^{-1/2} A D^{-1/2}$

$$g_\theta * x \approx \theta (I_N + D^{-1/2} A D^{-1/2}) x$$

- Applying ***renormalisation trick:***

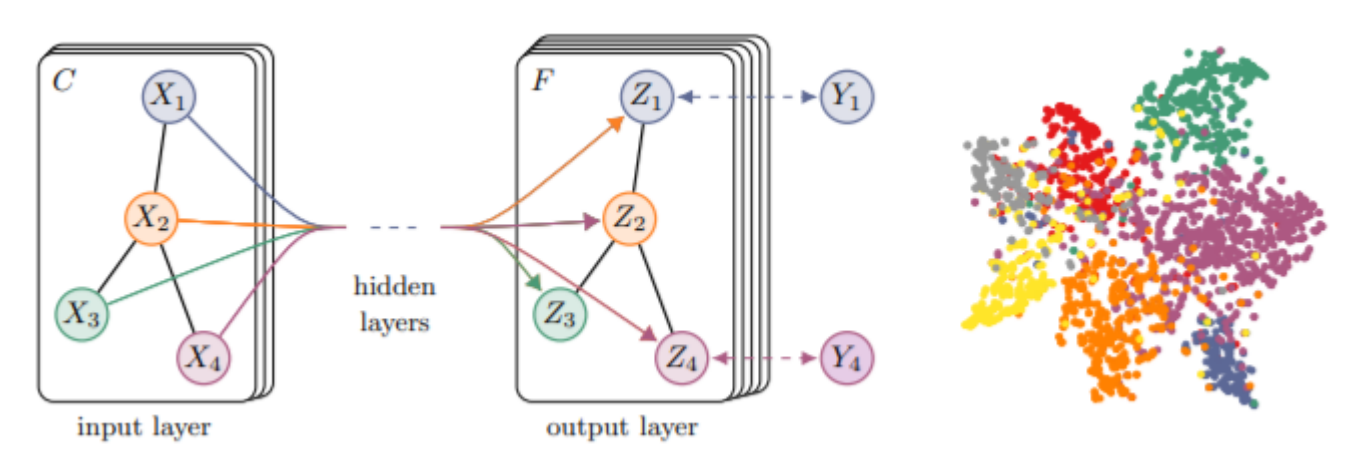
$$I_N + D^{-1/2} A D^{-1/2} \rightarrow \check{D}^{-1/2} \check{A} \check{D}^{-1/2} \quad \text{where } \check{A} = A + I_N \text{ and } \check{D}_{ii} = \sum_j \check{A}_{ij}$$

$$\mathbf{Z = \check{D}^{-1/2} \check{A} \check{D}^{-1/2} X \theta}$$

Theory?

- Connected nodes in the graph are likely to share the same label.
- Node does not include its own features : Ã=A+I
- Normalising feature values, using degree matrix.

- C : input channels
- F : feature maps
- Y : labels
- Colour : document classes



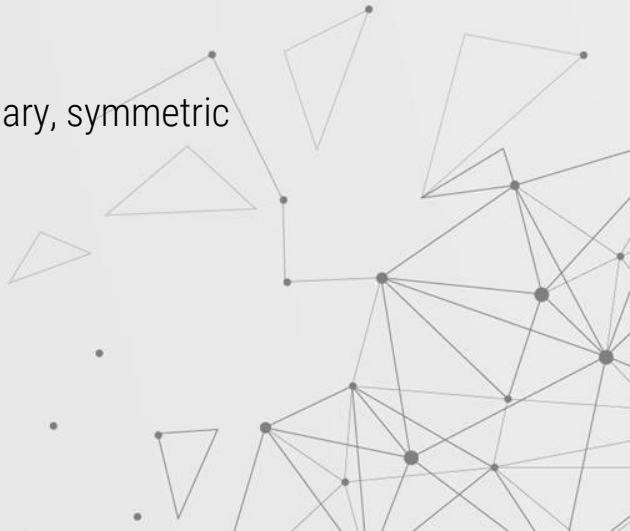
Data?

- In our study, we utilize two datasets, mainly **Pubmed** and **Cora**, both are citation datasets for scientific publications.

- The datasets contain sparse bag-of-words feature vectors for each document and a list of citation links between documents. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary.

- We treat the citation links as (undirected) edges and constructa binary, symmetric adjacency matrix A.
- Each document has a class label.

- N by N adjacency matrix (N is the number of nodes),
- N by D feature matrix (D is the number of features per node), and
- N by E binary label matrix (E is the number of classes).



Data Files?

- **x :** the feature vectors of the training instances
- **y :** the one-hot labels of the training instances,
- **graph :** a dict in the format {index: [index_of_neighbor_nodes]}, where the neighbor nodes are organized as a list.
- **allx :** the feature vectors of both labeled and unlabeled training instances (a superset of x)
- **tx :** the feature vectors of the test instances
- **ty :** the one-hot labels of the test instances
- **test.index :** the indices of test instances in graph
- **ally :** the labels for instances in allx

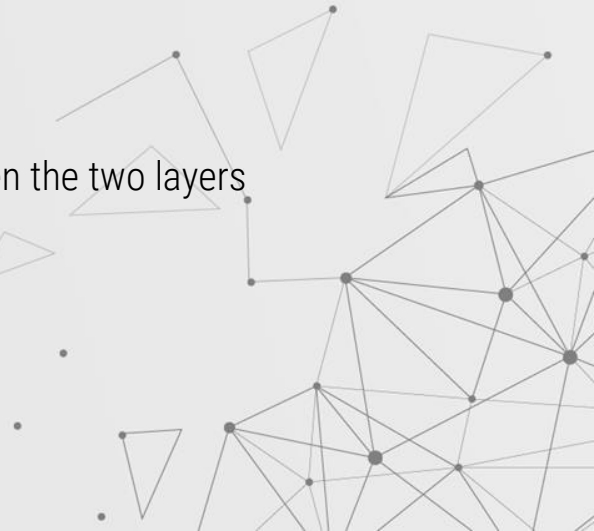
Dataset	Type	Nodes	Edges	Classes	Features
Cora	Citation network	2,708	5,429	7	1,433
Pubmed	Citation network	19,717	44,338	3	500

Data Processing?

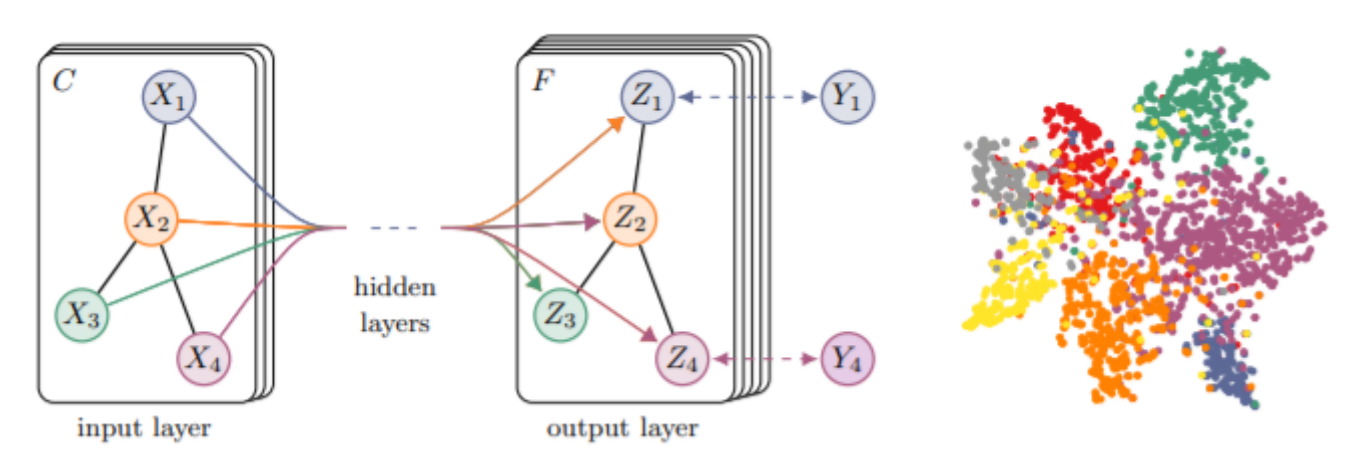
- **Normalize_features :** Normalize the data so that when the GCN layers are stacked, the node feature propagate in a smooth manner without explosion or vanishing.
- **Preprocess_adj :** This function returns $\hat{A}=\check{D}^{-½}\check{A}\check{D}^{-½}$ as a coordinate matrix
- **Sparse_mx_to_torch_sparse_tensor :** Returns a tensor containing the indices, values and shape of the input sparse matrix

Model?

1. **Parameter Initialization :** We applied Xavier Initialization for weights and biases by normalizing between the ranges of -sigma and +sigma where sigma = $1/\sqrt{N}$
1. **Layers :** Two layer model
1. **Hidden Units:** 16
1. **Dropouts :** One dropout with probability of 0.5 applied between the two layers
1. **Output :** $Z=f(X,A) = softmax(\hat{A}ReLU(\hat{A}XW^{0})W^{1})$



Model?



$$Z=f(X,A) = \text{softmax}(\hat{A}\text{ReLU}(\hat{A}XW^{0})W^{1})$$

where $\hat{A}=\check{D}^{-\frac{1}{2}}\check{A}\check{D}^{-\frac{1}{2}}$

Quantitative Results?

- The original implementation had the following accuracies:

Method	Cora	Pubmed
GCN (this paper)	**81.5** (4s)	**79.0** (38s)
GCN (rand. splits)	80.1 ± 0.5	78.9 ± 0.7

- We implemented accuracy as total_corect/total_labeled and used negative log likelihood loss. We managed to get an accuracy within 1% for both datasets.

CORA:

```
Test set results: loss= 0.6786 accuracy= 0.8060
```

PUBMED:

```
Test set results: loss= 0.5766 accuracy= 0.7840
```

- We used the same train/test splits and same sets of hyperparameters:
0.5 (dropout rate), $5x10^{-4}$(λ-L2 regularization), 16(number of hidden units), 200(epochs)

Qualitative Results?

- The output preds are class labels. 7 classes for cora, 3 for pubmed.

CORA

```
preds =  tensor([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 6, 5, 4, 4, 4, 1, 1, 1, 1,
        1, 1, 6, 2, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5,
        5, 5, 5, 5, 5, 2, 2, 2, 2, 2, 6, 6, 3, 0, 0, 0, 0, 5, 0, 0, 0, 3, 0, 0,
        6, 0, 6, 3])
labels =  tensor([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 2, 2, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 3, 4, 4, 4, 4, 1, 1, 3, 1,
        0, 3, 0, 2, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5,
        5, 5, 5, 5, 5, 2, 2, 2, 2, 1, 6, 6, 3, 0, 0, 5, 0, 5, 0, 3, 5, 3, 0, 0,
        6, 0, 6, 3])
```

PUBMED

```
preds =  tensor([2, 1, 0, 1, 0, 1, 1, 1, 2, 2, 1, 2, 2, 1, 0, 2, 1, 2, 0, 0, 1, 1, 1, 0,
        1, 1, 0, 1, 1, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 0, 1, 2, 1,
        1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 0, 1, 1, 1, 1, 1, 1, 2, 0, 2, 2, 2, 0,
        0, 1, 2, 1, 0, 0, 2, 0, 1, 2, 1, 0, 2, 2, 2, 2, 2, 1, 1, 1, 0, 1, 2, 2,
        1, 1, 0, 1])
labels =  tensor([1, 1, 1, 1, 0, 0, 1, 2, 2, 2, 1, 2, 2, 1, 0, 2, 2, 2, 0, 0, 2, 0, 1, 2,
        1, 1, 0, 1, 2, 0, 1, 2, 2, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 0, 1, 2, 1,
        1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 1, 1, 2, 0, 2, 2, 2, 2,
        0, 1, 1, 1, 0, 0, 2, 2, 2, 2, 1, 0, 2, 2, 2, 2, 2, 1, 1, 1, 0, 2, 2, 2,
        1, 1, 0, 1])
```

Additional Inferences

- The model works considering K=1 in chebyshev model. K^{th} order term accounts for neighbours at distance K from node.
- An important feature of T_k functions is they are related kth power of $T_k(\tilde{L})$. L is calculated using adjacency matrix of graph.
- The model works just like a 3x3 filter in convolution networks. It trains for weights on its own value(K=0) and immediate neighbour(K=1).
- The restricted linear model makes computations pretty fast and easy and also prevents overfitting on datasets(less parameters).

Takeaways?

- Working with PyTorch.
- Practical implementation of neural networks.
- Understanding about graphs, convolutions and different present models.

THANKS

Does anyone have any questions?