Semi-Supervised Classification with Graph Convolutional Networks

Presented by Shazia Akbar

Journal Club: 1st April 2019

Euclidean ConvNets

Local

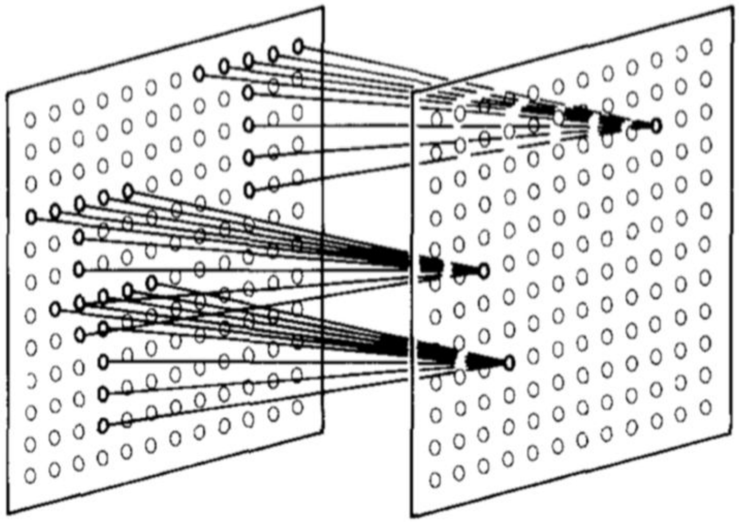
- Information encoded from local space

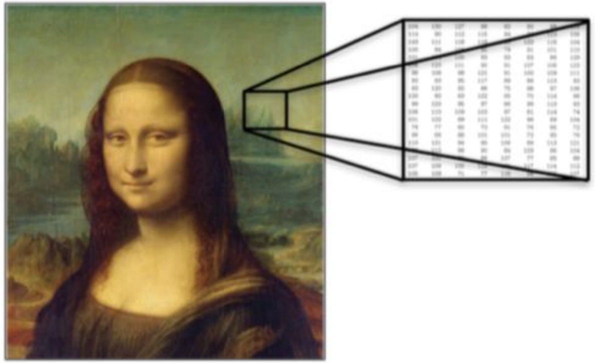
Stationary

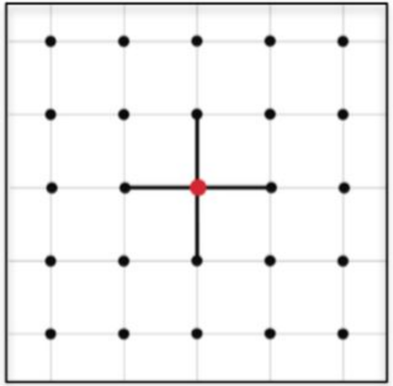
- Can recognize the same patch regardless of location

Multiscale

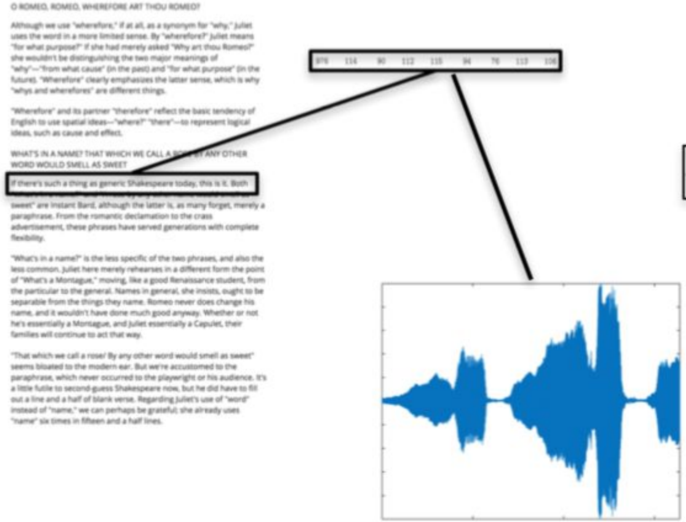
- Simple structures combined to make more abstract/complex structures







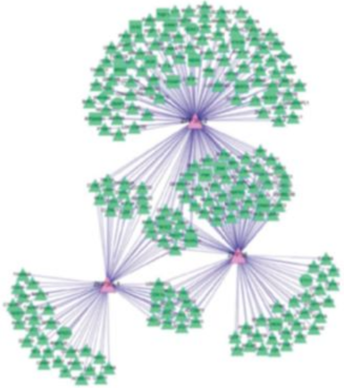
2D grids



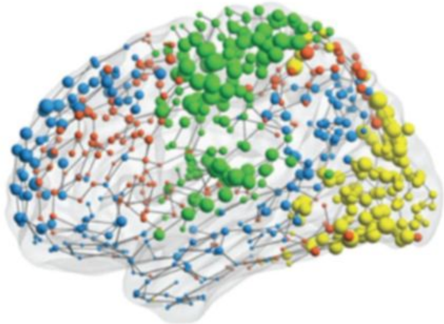
1D grid

Slides from Xavier Bresson

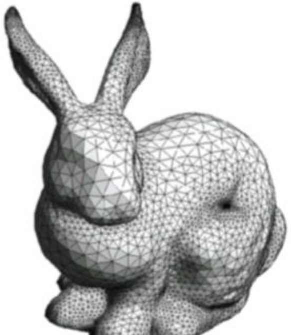
Social networks



Regulatory networks

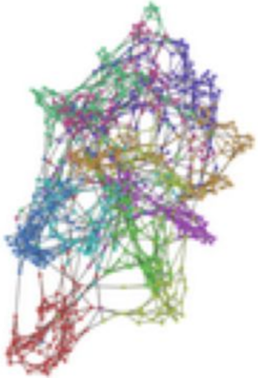


Functional networks



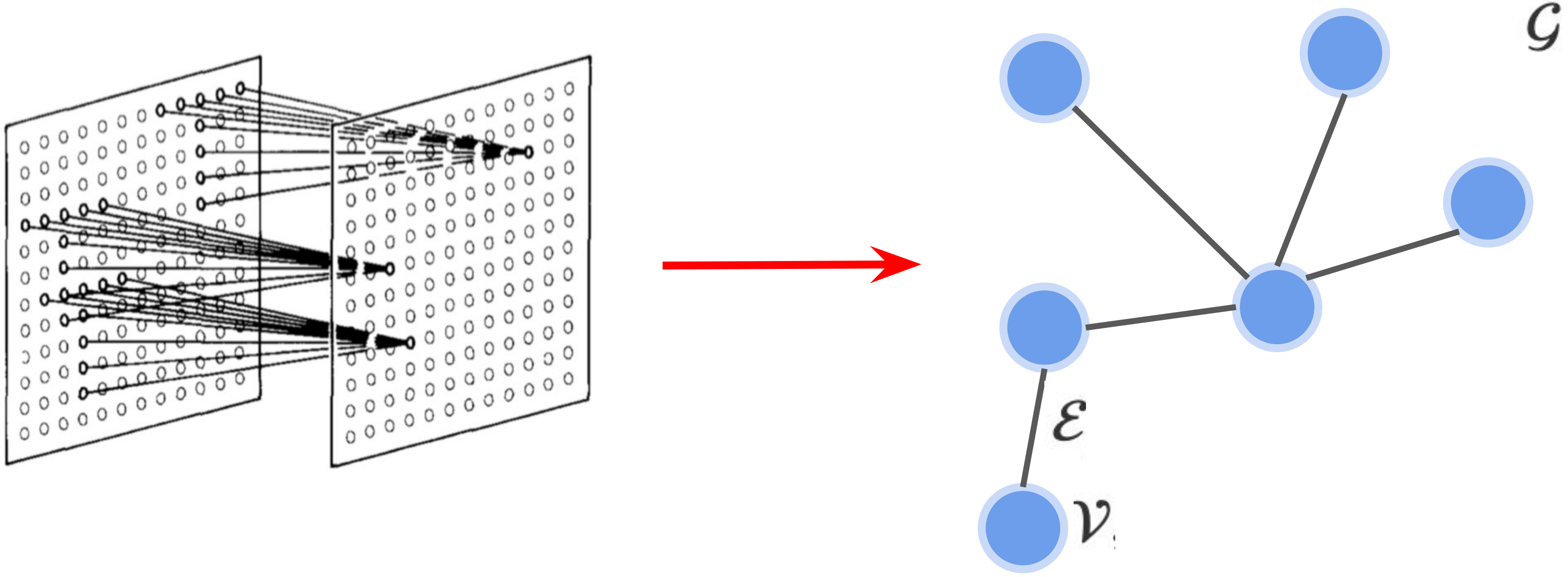
3D shapes

=



Graphs/
Networks

Slides from Xavier Bresson


$\mathcal{G}$
$\mathcal{E}$
$\mathcal{V}$

Graph Notation

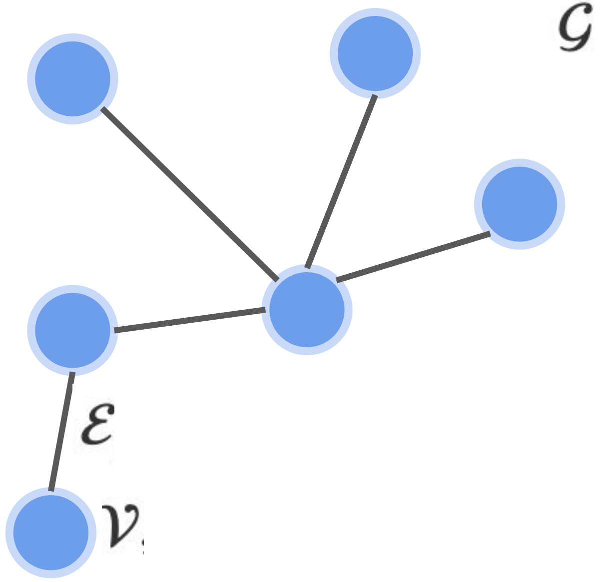
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$

N = Number of nodes (N=9)

Each node has feature vector, *f_i*

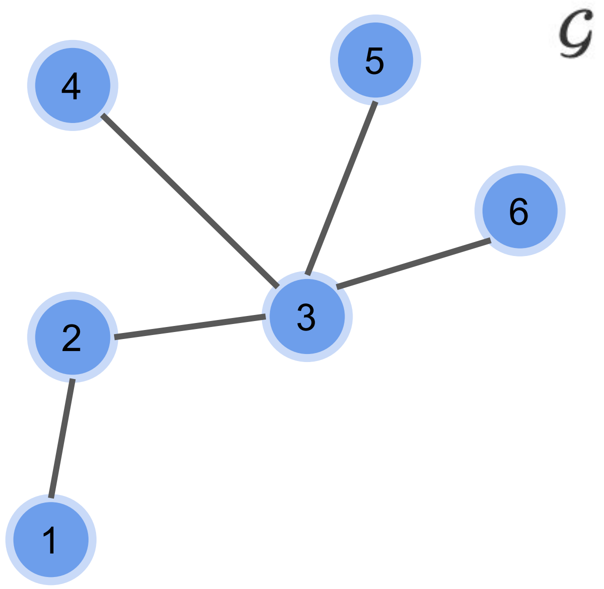
A = Adjacency matrix i.e. graph representation in binary form

- N x N matrix



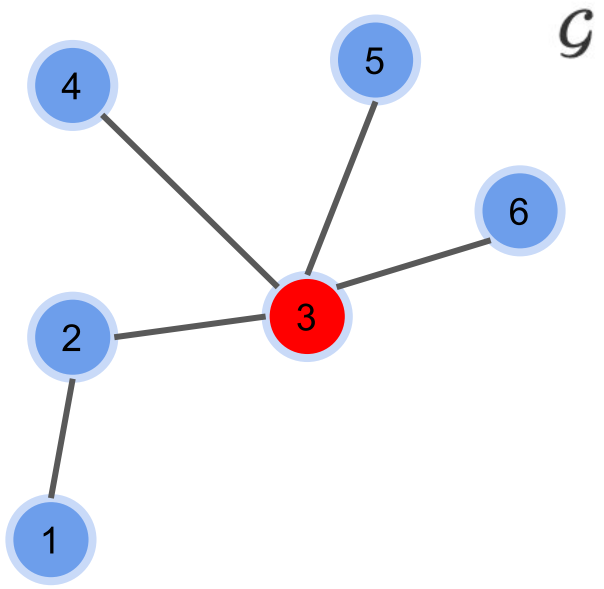
Adjacency Matrix *(A)*

0	1	0	0	0	0
1	0	1	0	0	0
0	1	0	1	1	1
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0

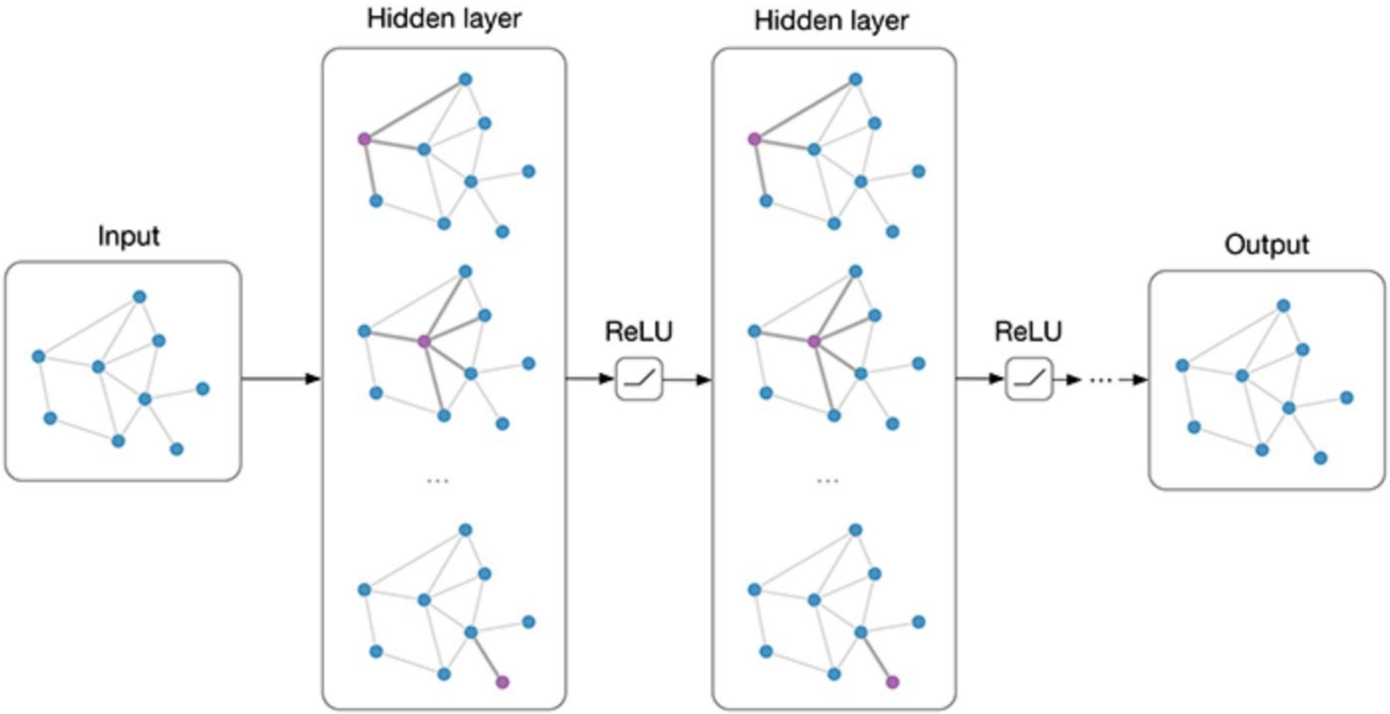


Adjacency Matrix (A)

0	1	0	0	0	0
1	0	1	0	0	0
0	**1**	**0**	**1**	**1**	**1**
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0



Graph ConvNet



Graph ConvNet

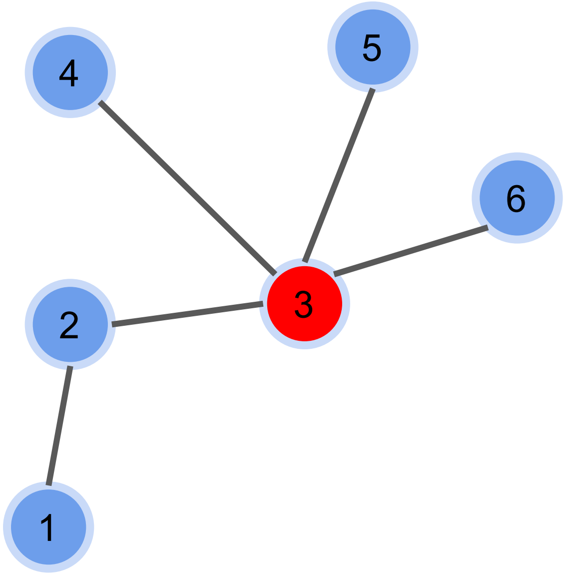
$$H^{(l+1)} = f(H^{(l)}, A),$$

ReLU

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right),$$

W are the weights in layer l

Graph ConvNet



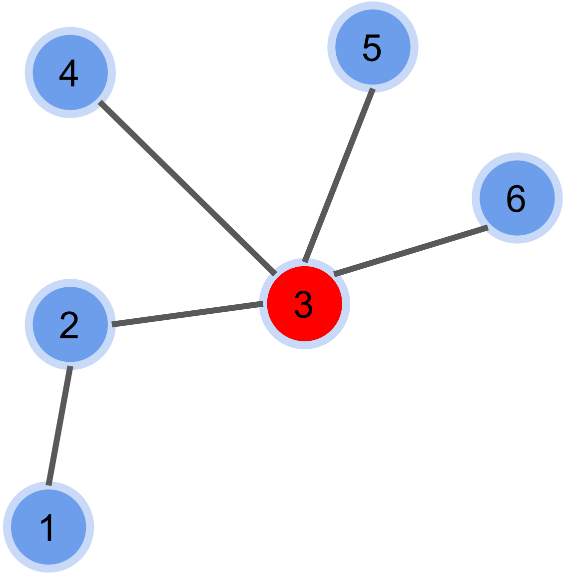
$$H^{(l+1)} = f(H^{(l)}, A) ,$$

ReLU

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right) ,$$

W are the weights in layer l

Graph ConvNet



$$H^{(l+1)} = f(H^{(l)}, A),$$

ReLU

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right),$$

W are the weights in layer l

$$A = A + I$$

Graph ConvNet

Layer-wise backprop rule

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right),$$

Diagonal Node Degree Matrix

(basically count of neighbours at each node)

Graph ConvNets Layers

Spectral Graph Convolutions

Filter parameterized by parameters in Fourier space:

$$(f \star g)_i = \underbrace{\sum_{k \geq 1} \underbrace{\langle f, \phi_k \rangle_{L^2(\mathcal{V})} \langle g, \phi_k \rangle_{L^2(\mathcal{V})}}_{\text{product in the Fourier domain}} \phi_{k,i}}_{\text{inverse Fourier transform}}$$

Expensive!

Approximate graph filters

We can approximate parameters of our filter using Chebyshev polynomials:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \,,$$

"It depends only on nodes that are at maximum K steps away from central node (Kth order neighborhood)"

Approximate graph filters

We can approximate parameters of our filter using Chebyshev polynomials:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \,,$$

Chebyshev polynomials which are iteratively defined

"It depends only on nodes that are at maximum K steps away from central node (Kth order neighborhood)"

Layer-wise Linear Model

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \, ,$$

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \, ,$$

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \, ,$$

...

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \, ,$$

Can stack these graph conv to gather abstraction

When K=1…

Layer-wise Linear Model

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,,$$

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,,$$

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,,$$

...

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,,$$

Can stack these graph conv to gather abstraction

When K=1…

- Prevent overfitting
- Build deeper models

$\lambda_{\max} \approx 2,$

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \, ,$$

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \, , \tag{6}$$

$\lambda_{\text{max}} \approx 2,$

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,,$$

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \,, \tag{6}$$

$$\lambda_{\max} \approx 2,$$

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \, ,$$

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \, , \tag{6}$$

$$g_{\theta} \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x \, , \tag{7}$$

After renormalization trick

Input data with C channels

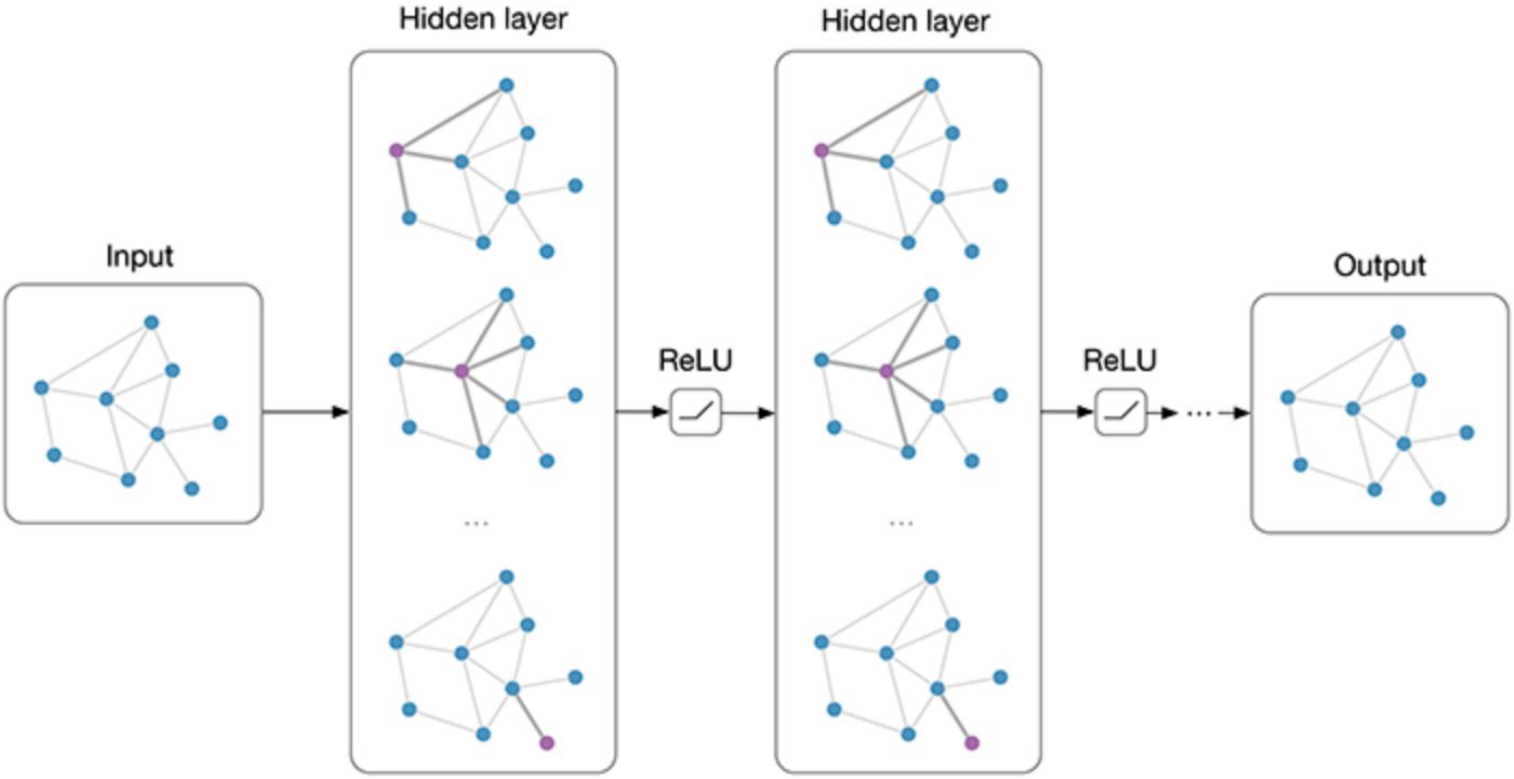
$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \,, \qquad (8)$$

Introduced with renormalization trick

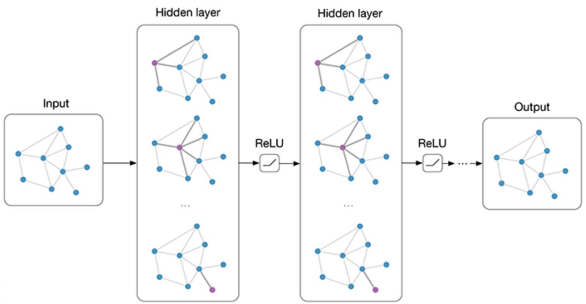
Filter parameters with CxF dimensions (F = no. of filters)

$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

Semi-supervised



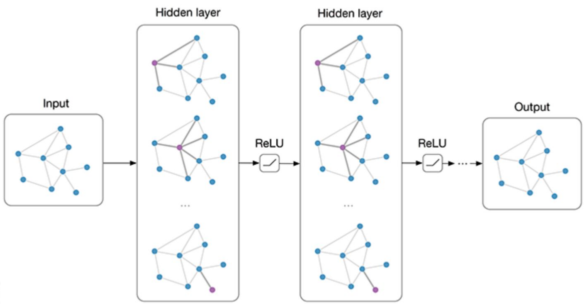
Semi-supervised



Forward pass:

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

Semi-supervised

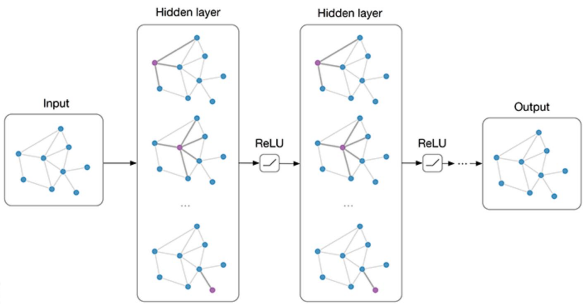


Forward pass:

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) W^{(1)}\right) .$$

Compute cross entropy on **known** labels

Semi-supervised



Forward pass:

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

Compute cross entropy on **known** labels

Backprop using gradient descent

(Need entire dataset to fit into memory)

Experiments

Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Results

Table 2: Summary of results in terms of classification accuracy (in percent).

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	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

Iterative bootstrapping with 2 regression classifiers

Performance of 10 randomly drawn splits

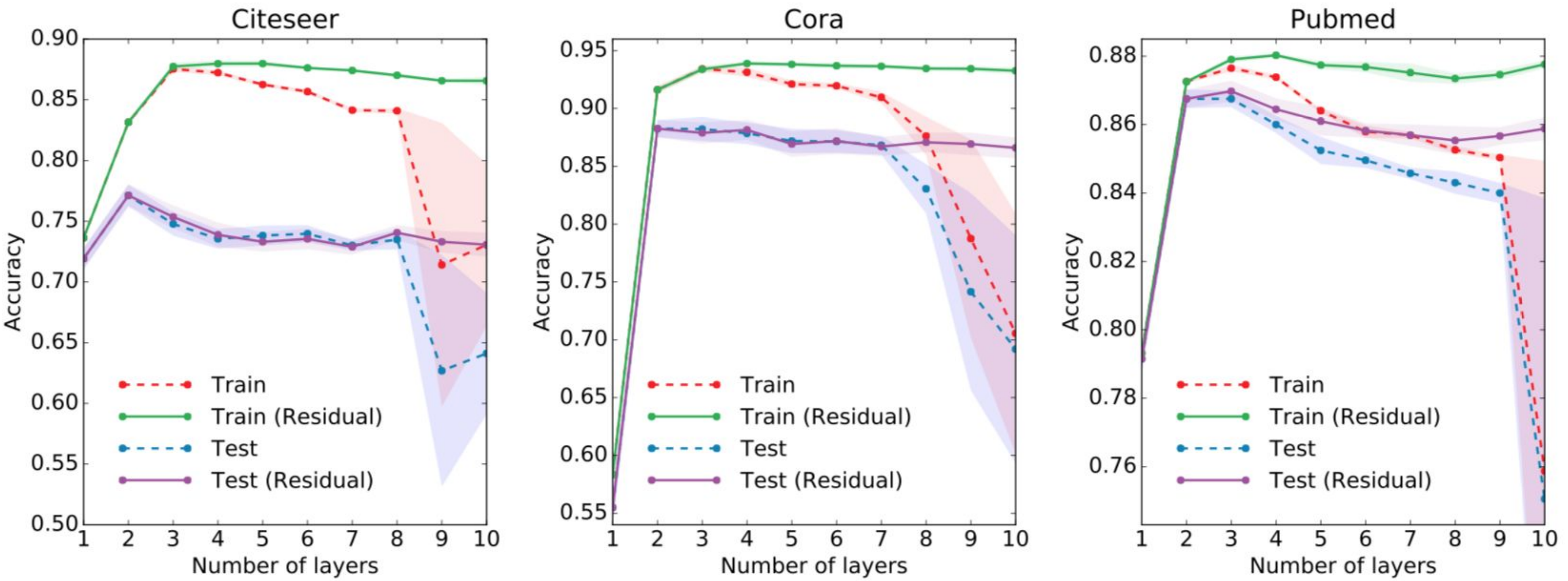
Results

Description		Propagation model	Citeseer	Cora	Pubmed
Chebyshev filter (Eq. 5)	$K = 3$	$\sum_{k=0}^{K} T_k(\tilde{L}) X \Theta_k$	69.8	79.5	74.4
	$K = 2$		69.6	81.2	73.8
1st-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) X \Theta$	69.3	79.2	77.4
Renormalization trick (Eq. 8)		$\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$	**70.3**	**81.5**	**79.0**
1st-order term only		$D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

More info from neighbours

Similar performance with fewer variables

Still performs fairly well



Citeseer
Cora
Pubmed
Accuracy
Number of layers
Train
Train (Residual)
Test
Test (Residual)

Results

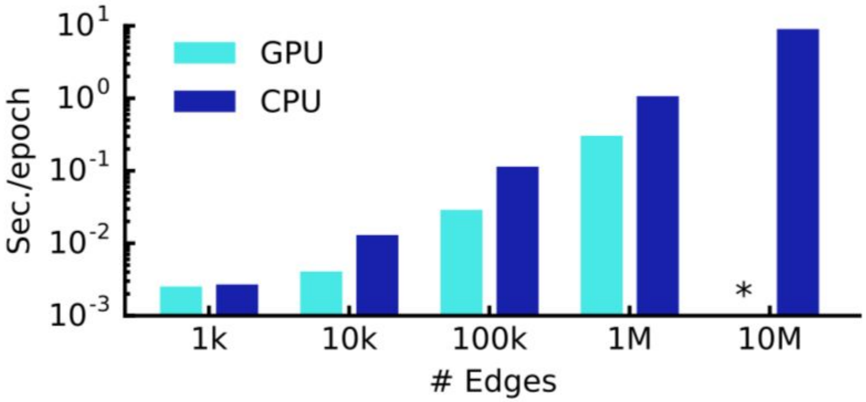


Figure 2: Wall-clock time per epoch for random graphs. (*) indicates out-of-memory error.

Overcome memory issue

Authors suggest mini-batch SGD

FastGCN (Monte-carlo): https://openreview.net/forum?id=rytstxWAW

PinSage (Random walks): https://cs.stanford.edu/~jure/pubs/pinsage-kdd18.pdf

Work since 2017...

“Exploiting edge features in Graph Neural Networks”:
https://arxiv.org/pdf/1809.02709.pdf

“MotifNet: A motif-based Graph Convolutional Network for directed graphs”:
https://arxiv.org/abs/1802.01572

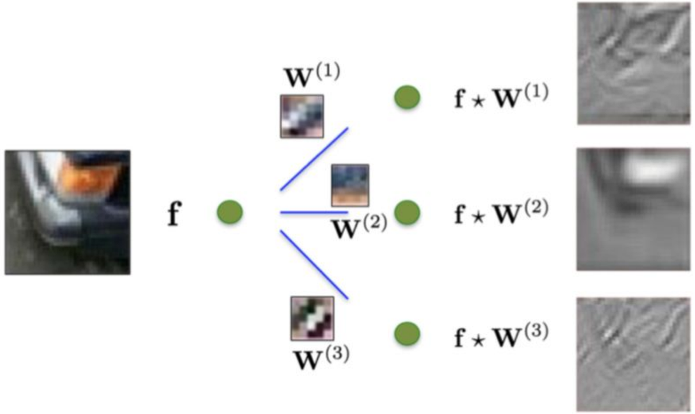
“A Comprehensive Survey on Graph Neural Networks”:
https://arxiv.org/pdf/1901.00596.pdf

Demo

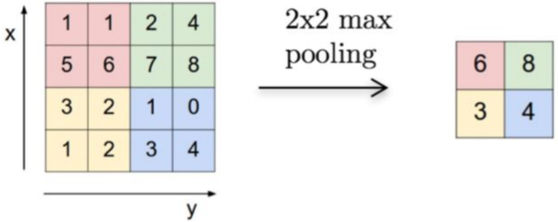
https://tkipf.github.io/graph-convolutional-networks/

- Locality: Compact support kernels $\Rightarrow O(1)$ parameters per filter.

- Stationarity: Convolutional operators $\Rightarrow O(n\log n)$ in general (FFT) and $O(n)$ for compact kernels.



- Multi-scale: Downsampling + pooling $\Rightarrow O(n)$



Slides from Xavier Bresson