

# Neural Networks on Graphs

Machine Learning

*Federico Magnolfi & Niccolò Biondi*

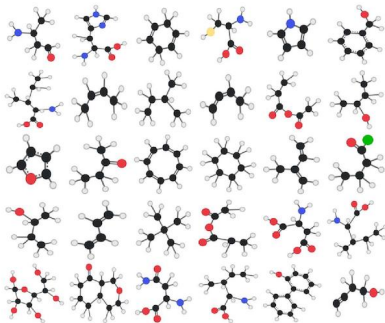
Prof. Paolo Frasconi

20 April 2020



# Overview

# Graphs

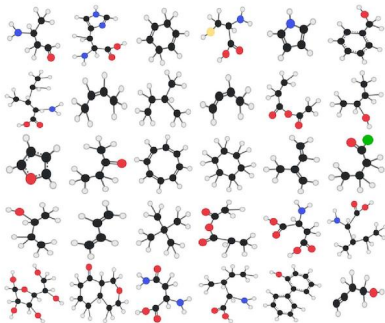


Many **application domains**



Many **tasks**

# Graphs



Many **application domains**

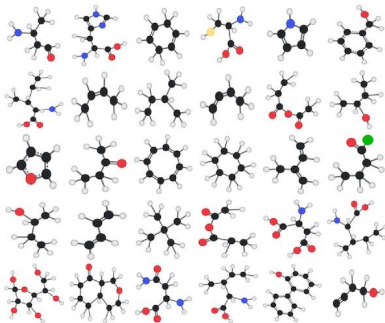
■ chemistry



Many **tasks**

■ graph classification

# Graphs



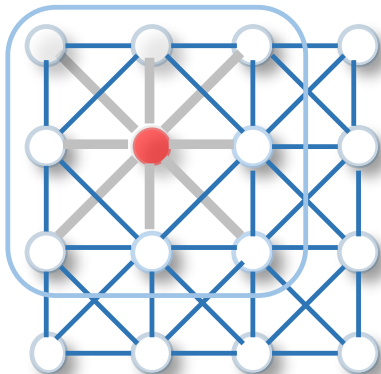
Many **application domains**

- chemistry
- social networks

Many **tasks**

- graph classification
- node classification

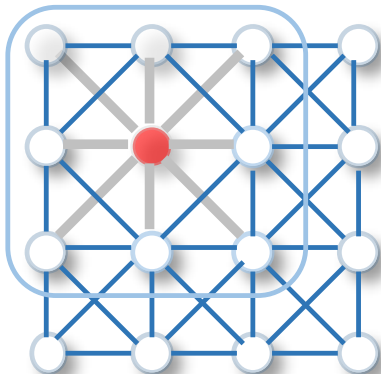
# Graphs: difficulties



Images

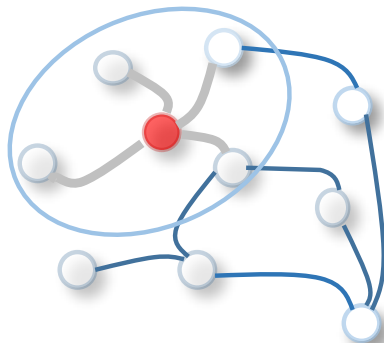
- **regular** structure
- **ordered** neighbors

# Graphs: difficulties



Images

- **regular** structure
- **ordered** neighbors



Graphs

- **irregular** structure
- **unordered** neighbors

# Some definitions

Graph:  $\mathcal{G} = (A, X)$

- Adjacency matrix:  $A$
- Feature matrix:  $X$



# Some definitions

Graph:  $\mathcal{G} = (A, X)$

- Adjacency matrix:  $A \in \mathbb{R}^{N \times N}$
- Feature matrix:  $X \in \mathbb{R}^{N \times F}$
- $N$  nodes,  $F$  features per node

# Some definitions

Graph:  $\mathcal{G} = (A, X)$

- Adjacency matrix:  $A \in \mathbb{R}^{N \times N}$
- Feature matrix:  $X \in \mathbb{R}^{N \times F}$
- $N$  nodes,  $F$  features per node
- Degree of node  $i$ :  $D_{ii} = \sum_j A_{ij}$
- Graph Laplacian:  $L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

# Some definitions

Graph:  $\mathcal{G} = (A, X)$

- Adjacency matrix:  $A \in \mathbb{R}^{N \times N}$
- Feature matrix:  $X \in \mathbb{R}^{N \times F}$
- $N$  nodes,  $F$  features per node
- Degree of node  $i$ :  $D_{ii} = \sum_j A_{ij}$
- Graph Laplacian:  $L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

Remark

$A$  sparse  $\Rightarrow L$  sparse

# Objective of this project

## Objective

**Reproduce** some **experimental results** of 4 papers  
on three different datasets

- 1 Planetoid (Yang et al., 2016)
- 2 ChebNet (Defferrard et al., 2016)
- 3 GCN (Kipf & Welling et al., 2017)
- 4 GAT (Veličković et al., 2018)

All these algorithms:

- use **Neural Networks**
- to **predict node class**
- in a **semi-supervised** setting

## Datasets: citation networks

	Cora	Citeseer	Pubmed
# Nodes	2708	3327	19717
# Edges	5429	4732	44338
# Features/Node	1433	3703	500
# Classes	7	6	3

# Datasets: citation networks

	Cora	Citeseer	Pubmed
# Nodes	2708	3327	19717
# Edges	5429	4732	44338
# Features/Node	1433	3703	500
# Classes	7	6	3

- Features: **bag of words**

# Datasets: citation networks

	Cora	Citeseer	Pubmed
# Nodes	2708	3327	19717
# Edges	5429	4732	44338
# Features/Node	1433	3703	500
# Classes	7	6	3
# Training Nodes	140	120	60
# Validation Nodes	500	500	500
# Test Nodes	1000	1000	1000

- Features: **bag of words**

# Chebbychev Networks

(Defferrard et al., 2016)



# Spectral Convolutional GNN

Assume the graph is undirected

# Spectral Convolutional GNN

Assume the graph is undirected



$A$  is symmetric

# Spectral Convolutional GNN

Assume the graph is undirected



$A$  is symmetric



$L$  is symmetric, real, and  $\geq 0$

# Spectral Convolutional GNN

Assume the graph is undirected



$A$  is symmetric



$L$  is symmetric, real, and  $\geq 0$



$$\exists U, \Lambda : L = U \Lambda U^T$$

$U$ : eigenvectors matrix

$\Lambda$ : eigenvalues matrix

# Spectral Convolutional GNN

Assume the graph is undirected



$A$  is symmetric



$L$  is symmetric, real, and  $\geq 0$



$$\exists U, \Lambda : L = U \Lambda U^T$$

$U$ : eigenvectors matrix

$\Lambda$ : eigenvalues matrix

Fourier Transform

$$\mathcal{F}(x) = U^T x$$

# Spectral Convolutional GNN

Assume the graph is undirected



$A$  is symmetric



$L$  is symmetric, real, and  $\geq 0$



$$\exists U, \Lambda : L = U \Lambda U^T$$

$U$ : eigenvectors matrix

$\Lambda$ : eigenvalues matrix

Fourier Transform

$$\mathcal{F}(x) = U^T x$$



Inverse Fourier Transform

$$\mathcal{F}^{-1}(x) = Ux$$

# Spectral Convolutional GNN

Assume the graph is undirected



$A$  is symmetric



$L$  is symmetric, real, and  $\geq 0$



$$\exists U, \Lambda : L = U \Lambda U^T$$

$U$ : eigenvectors matrix

$\Lambda$ : eigenvalues matrix

Fourier Transform

$$\mathcal{F}(x) = U^T x$$



Inverse Fourier Transform

$$\mathcal{F}^{-1}(x) = Ux$$



Convolution

$$\begin{aligned} g * x &= U(U^T g \odot U^T x) \\ &= U g_{\theta} U^T x \end{aligned}$$

# Spectral Convolutional GNN

Assume the graph is undirected



$A$  is symmetric



$L$  is symmetric, real, and  $\geq 0$



$$\exists U, \Lambda : L = U\Lambda U^T$$

$U$ : eigenvectors matrix

$\Lambda$ : eigenvalues matrix

Fourier Transform

$$\mathcal{F}(x) = U^T x$$



Inverse Fourier Transform

$$\mathcal{F}^{-1}(x) = Ux$$



Convolution

$$\begin{aligned} g * x &= U(U^T g \odot U^T x) \\ &= Ug_{\theta} U^T x \end{aligned}$$

## Spectral Convolutional GNN

Spectral ConvGNNs follow this definition and differ in the choice of  $g_{\theta}$



# Spectral Convolutional GNN

Assume the graph is undirected



$A$  is symmetric



$L$  is symmetric, real, and  $\geq 0$



$$\exists U, \Lambda : L = U \Lambda U^T$$

$U$ : eigenvectors matrix

$\Lambda$ : eigenvalues matrix

Fourier Transform

$$\mathcal{F}(x) = U^T x$$



Inverse Fourier Transform

$$\mathcal{F}^{-1}(x) = Ux$$



Convolution

$$\begin{aligned} g * x &= U(U^T g \odot U^T x) \\ &= U g_{\theta} U^T x \end{aligned}$$

## Spectral Convolutional GNN

Spectral ConvGNNs follow this definition and differ in the choice of  $g_{\theta}$

## Problems

**1**  $N$  parameters per filter

**2** filters not localized in space

# Chebyshev Networks (ChebNets)

## ChebNet

Spectral ConvGNN that obtains  $g_\theta$  from **Chebyshev polynomials** of  $\tilde{\Lambda}$

# Chebyshev Networks (ChebNets)

## ChebNet

Spectral ConvGNN that obtains  $g_\theta$  from **Chebyshev polynomials** of  $\tilde{\Lambda}$

$$T_k(Z) = \begin{cases} I_N, & \text{if } k = 0 \\ Z, & \text{if } k = 1 \\ 2ZT_{k-1}(Z) - T_{k-2}(Z), & \text{if } k > 1 \end{cases}$$

# Chebyshev Networks (ChebNets)

## ChebNet

Spectral ConvGNN that obtains  $g_\theta$  from **Chebyshev polynomials** of  $\tilde{\Lambda}$

$$T_k(Z) = \begin{cases} I_N, & \text{if } k = 0 \\ Z, & \text{if } k = 1 \\ 2ZT_{k-1}(Z) - T_{k-2}(Z), & \text{if } k > 1 \end{cases}$$

$$\tilde{\Lambda} = 2\Lambda/\lambda_{\max} - I_n$$

$$\tilde{L} = 2L/\lambda_{\max} - I_n$$

# Chebyshev Networks (ChebNets)

## ChebNet

Spectral ConvGNN that obtains  $g_\theta$  from **Chebyshev polynomials** of  $\tilde{\Lambda}$

$$T_k(Z) = \begin{cases} I_N, & \text{if } k = 0 \\ Z, & \text{if } k = 1 \\ 2ZT_{k-1}(Z) - T_{k-2}(Z), & \text{if } k > 1 \end{cases}$$

$$g_\theta \triangleq \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$$

$$\tilde{\Lambda} = 2\Lambda / \lambda_{\max} - I_n$$

$$\tilde{L} = 2L / \lambda_{\max} - I_n$$

# Chebyshev Networks (ChebNets)

## ChebNet

Spectral ConvGNN that obtains  $g_\theta$  from **Chebyshev polynomials** of  $\tilde{\Lambda}$

$$T_k(Z) = \begin{cases} I_N, & \text{if } k = 0 \\ Z, & \text{if } k = 1 \\ 2ZT_{k-1}(Z) - T_{k-2}(Z), & \text{if } k > 1 \end{cases}$$

$$\tilde{\Lambda} = 2\Lambda/\lambda_{\max} - I_n$$

$$\tilde{L} = 2L/\lambda_{\max} - I_n$$

$$g_\theta \triangleq \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$$

$$g_\theta * x = U \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) U^T x$$

# Chebyshev Networks (ChebNets)

## ChebNet

Spectral ConvGNN that obtains  $g_\theta$  from **Chebyshev polynomials** of  $\tilde{\Lambda}$

$$T_k(Z) = \begin{cases} I_N, & \text{if } k = 0 \\ Z, & \text{if } k = 1 \\ 2ZT_{k-1}(Z) - T_{k-2}(Z), & \text{if } k > 1 \end{cases}$$

$$\tilde{\Lambda} = 2\Lambda/\lambda_{\max} - I_n$$

$$\tilde{L} = 2L/\lambda_{\max} - I_n$$

$$g_\theta \triangleq \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$$

$$g_\theta * x = U \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) U^T x$$

$$UT_k(\tilde{\Lambda})U^T = T_k(\tilde{L})$$

# Chebyshev Networks (ChebNets)

## ChebNet

Spectral ConvGNN that obtains  $g_\theta$  from **Chebyshev polynomials** of  $\tilde{\Lambda}$

$$T_k(Z) = \begin{cases} I_N, & \text{if } k = 0 \\ Z, & \text{if } k = 1 \\ 2ZT_{k-1}(Z) - T_{k-2}(Z), & \text{if } k > 1 \end{cases}$$

$$g_\theta \triangleq \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$$

$$g_\theta * x = U \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) U^T x$$

$$\tilde{\Lambda} = 2\Lambda / \lambda_{\max} - I_n$$

$$\tilde{L} = 2L / \lambda_{\max} - I_n$$

$$UT_k(\tilde{\Lambda})U^T = T_k(\tilde{L})$$

$$g_\theta * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$



# Chebyshev layer

$$\theta \in \mathbb{R}^K$$

$$T_k(\tilde{L}) \in \mathbb{R}^{N \times N}$$

$$x \in \mathbb{R}^N$$

Single-filter, single-feature

$$g_{\theta} * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

# Chebyshev layer

$$\theta \in \mathbb{R}^K$$

$$\Theta \in \mathbb{R}^{K \times F_{in} \times F_{out}}$$

$$T_k(\tilde{L}) \in \mathbb{R}^{N \times N}$$

$$x \in \mathbb{R}^N$$

$$X \in \mathbb{R}^{N \times F_{in}}$$

## Single-filter, single-feature

$$g_{\theta} * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

# Chebyshev layer

$$\theta \in \mathbb{R}^K$$

$$\Theta \in \mathbb{R}^{K \times F_{in} \times F_{out}}$$

$$T_k(\tilde{L}) \in \mathbb{R}^{N \times N}$$

$$x \in \mathbb{R}^N$$

$$X \in \mathbb{R}^{N \times F_{in}}$$

## Single-filter, single-feature

$$g_{\theta} * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

## Many filters, many features

$$G_{\Theta} * X = \sum_{k=0}^{K-1} T_k(\tilde{L})X\Theta_k$$

- $T_k$ : cheb polynomial of order  $k$
- $X$ : input
- $\Theta$ : learnable weights

## Chebyshev layer: observations

$$G_{\Theta} * X = \sum_{k=0}^{K-1} T_k(\tilde{L}) X \Theta_k$$

### Sparsity

$A$  is sparse  $\Rightarrow L, \tilde{L}$  are sparse  $\Rightarrow \{T_k(\tilde{L})\}_k$  are sparse in practice

## Chebyshev layer: observations

$$G_{\Theta} * X = \sum_{k=0}^{K-1} T_k(\tilde{L}) X \Theta_k$$

### Sparsity

$A$  is sparse  $\Rightarrow L, \tilde{L}$  are sparse  $\Rightarrow \{T_k(\tilde{L})\}_k$  are sparse in practice

### Considerations

- $T_k(\tilde{L})X$ : is a sparse-dense matmul
- $[T_k(\tilde{L})X]\Theta_k$ : is a dense matmul
- efficiently computed on parallel architectures

# Graph Convolutional Networks

(Kipf & Welling, 2017)

# Graph Convolutional Networks

## Graph Convolutional Networks

**GCNs** are a first-order approximations of ChebNet, i.e.  $K=2$ .

# Graph Convolutional Networks

## Graph Convolutional Networks

**GCNs** are a first-order approximations of ChebNet, i.e.  $K=2$ .

- approximate also:  $\lambda_{max} \simeq 2$
- add the constraint:  $\theta_0 = -\theta_1 \triangleq \theta$



# Graph Convolutional Networks

## Graph Convolutional Networks

**GCNs** are a first-order approximations of ChebNet, i.e.  $K=2$ .

- approximate also:  $\lambda_{max} \simeq 2$
- add the constraint:  $\theta_0 = -\theta_1 \triangleq \theta$

### ChebNet convolution

$$g_{\theta} * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

# Graph Convolutional Networks

## Graph Convolutional Networks

**GCNs** are a first-order approximations of ChebNet, i.e.  $K=2$ .

- approximate also:  $\lambda_{max} \simeq 2$
- add the constraint:  $\theta_0 = -\theta_1 \triangleq \theta$

### ChebNet convolution

$$g_{\theta} * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

### GCN convolution

$$g_{\theta} * x = \theta(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})x$$

# Graph Convolutional Networks

## Graph Convolutional Networks

**GCNs** are a first-order approximations of ChebNet, i.e.  $K=2$ .

- approximate also:  $\lambda_{max} \simeq 2$
- add the constraint:  $\theta_0 = -\theta_1 \triangleq \theta$

### ChebNet convolution

$$g_{\theta} * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

### GCN convolution

$$g_{\theta} * x = \theta(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})x$$

### Instability

eigenvalues of  $(\cdot) \in [0, 2]$

# Graph Convolutional Networks

## Graph Convolutional Networks

**GCNs** are a first-order approximations of ChebNet, i.e.  $K=2$ .

- approximate also:  $\lambda_{max} \simeq 2$
- add the constraint:  $\theta_0 = -\theta_1 \triangleq \theta$

### ChebNet convolution

$$g_{\theta} * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

### GCN convolution

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

### Renormalization trick

$$g_{\theta} * x = \theta(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}})x$$

### Instability

eigenvalues of  $(\cdot) \in [0, 2]$

# Graph Convolutional Networks

## Graph Convolutional Networks

**GCNs** are a first-order approximations of ChebNet, i.e.  $K=2$ .

- approximate also:  $\lambda_{max} \simeq 2$
- add the constraint:  $\theta_0 = -\theta_1 \triangleq \theta$

### ChebNet convolution

$$g_{\theta} * x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

### GCN convolution

$$g_{\theta} * x = \theta(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})x$$

### Instability

eigenvalues of  $(\cdot) \in [0, 2]$

### Renormalization trick

$$g_{\theta} * x = \theta(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})x$$

### Definition

$$M \triangleq \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$$

# Graph Convolutional Layer

$$\theta \in \mathbb{R}$$

$$M \in \mathbb{R}^{N \times N}$$

$$x \in \mathbb{R}^N$$

Single-filter, single-feature

$$g_{\theta} * x = \theta Mx$$

# Graph Convolutional Layer

$$\theta \in \mathbb{R}$$

$$\Theta \in \mathbb{R}^{F_{in} \times F_{out}}$$

$$M \in \mathbb{R}^{N \times N}$$

$$x \in \mathbb{R}^N$$

$$X \in \mathbb{R}^{N \times F_{in}}$$

Single-filter, single-feature

$$g_{\theta} * x = \theta Mx$$

# Graph Convolutional Layer

$$\theta \in \mathbb{R}$$

$$\Theta \in \mathbb{R}^{F_{in} \times F_{out}}$$

$$M \in \mathbb{R}^{N \times N}$$

$$x \in \mathbb{R}^N$$

$$X \in \mathbb{R}^{N \times F_{in}}$$

## Single-filter, single-feature

$$g_{\theta} * x = \theta Mx$$

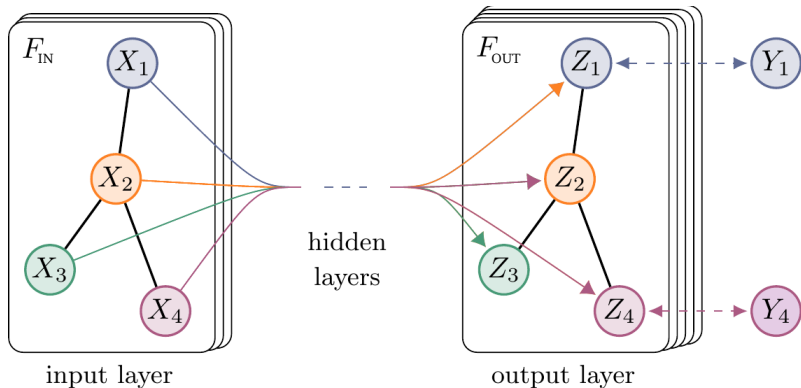
## Many filters, many features

$$G_{\Theta} * X = MX\Theta$$

- $M$ : renormalized matrix
- $X$ : input
- $\Theta$ : learnable weights



# Graph Convolutional Layer

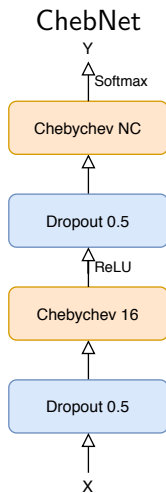


# Experimental Setup

## Preprocessing

- row-normalize  $X$

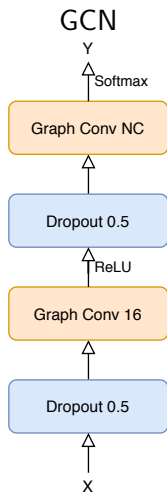
# Experimental Setup



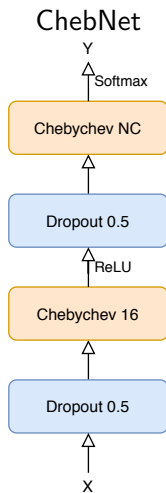
Preprocessing

■ row-normalize  $X$

Models



# Experimental Setup

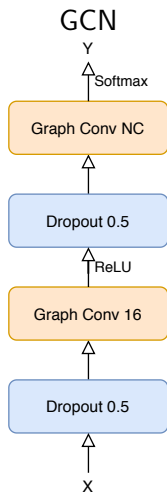


## Preprocessing

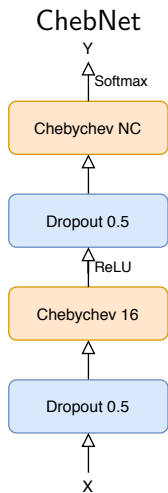
- row-normalize  $X$

## Models

- loss: *categorical cross-entropy*
- regularization: L2 on first layer
- hyperparameters: selected with GCN on Cora



# Experimental Setup



## Preprocessing

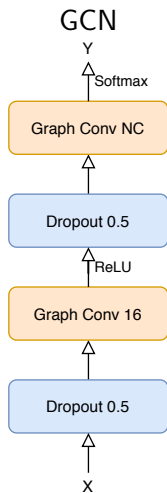
- row-normalize  $X$

## Models

- loss: *categorical cross-entropy*
- regularization: L2 on first layer
- hyperparameters: selected with GCN on Cora

## Evaluation

- metric: accuracy on test nodes



# Results for ChebNet and GCN

Method	Cora	Citeseer	Pubmed
<b>ChebNet</b>	81.2%	69.8%	74.4%
<b>ChebNet*</b>	82.0 $\pm$ 0.6%	70.5 $\pm$ 0.7%	75.2 $\pm$ 1.8%

\*: (our) 100 runs, Yang data split, *optimization std*

# Results for ChebNet and GCN

Method	Cora	Citeseer	Pubmed
<b>ChebNet</b>	81.2%	69.8%	74.4%
<b>ChebNet*</b>	82.0 $\pm$ 0.6%	70.5 $\pm$ 0.7%	75.2 $\pm$ 1.8%
<b>GCN</b>	81.5%	70.3%	79.0%
<b>GCN*</b>	80.6 $\pm$ 0.6%	68.7 $\pm$ 0.9%	78.3 $\pm$ 0.5%

\*: (our) 100 runs, Yang data split, *optimization std*

# Results for ChebNet and GCN

Method	Cora	Citeseer	Pubmed
<b>ChebNet</b>	81.2%	69.8%	74.4%
<b>ChebNet*</b>	82.0 $\pm$ 0.6%	70.5 $\pm$ 0.7%	75.2 $\pm$ 1.8%
<b>GCN</b>	81.5%	70.3%	79.0%
<b>GCN*</b>	80.6 $\pm$ 0.6%	68.7 $\pm$ 0.9%	78.3 $\pm$ 0.5%
<b># Features</b>	1433	3703	500
<b># Classes</b>	7	6	3

\*: (our) 100 runs, Yang data split, *optimization std*



# Planetoid

(Yang et al., 2016)

# Some definitions

## Assumption

A node and its neighbors (**instance** and its **graph context**) tend to have same labels.

# Some definitions

## Assumption

A node and its neighbors (**instance** and its **graph context**) tend to have same labels.

## Semi-supervised framework loss

$$\mathcal{L} = \mathcal{L}_s + \lambda \mathcal{L}_u$$

$$\mathcal{L}_s = - \sum_{i=0}^N \sum_{j=0}^K y_{ij} \log(\hat{y}_{ij})$$

$\mathcal{L}_u \Rightarrow$  Embedding vs Laplacian regularization

# Planetoid Contributions

- Embeddings exploited on several points of view:
  - are incorporated into graph **semi-supervised task**
  - to learn the **distributional context** of instances
  - trained to **predict** class label and graph context of nodes

# Planetoid Contributions

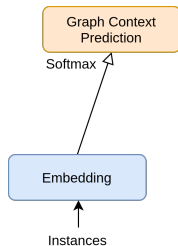
- Embeddings exploited on several points of view:
  - are incorporated into graph **semi-supervised task**
  - to learn the **distributional context** of instances
  - trained to **predict** class label and graph context of nodes

## Proposed Frameworks

- Planetoid-T  $\Rightarrow$  Transductive
- Planetoid-I  $\Rightarrow$  Inductive

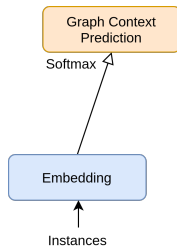
# Embeddings

- Softmax layer on embeddings w.r.t. all nodes
- *Similar* instances close in embedding space
  - nearby nodes in the graph
  - nodes with same labels



# Embeddings

- Softmax layer on embeddings w.r.t. all nodes
- *Similar* instances close in embedding space
  - nearby nodes in the graph
  - nodes with same labels



## Embedding loss

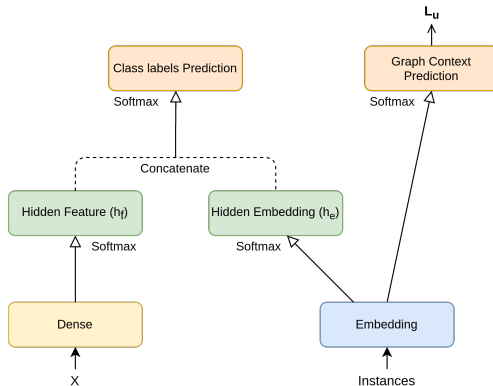
Embedding learning methods minimize the **log loss** of predicting context from the embedding of an instance (Skipgram model)

$$\mathcal{L}_u = - \sum_{(i,c)} \log p(c|i) = - \sum_{(i,c)} \log \left( \frac{\exp(\mathbf{w}_c^T \mathbf{e}_i)}{\sum_{c' \in \mathcal{C}} \exp(\mathbf{w}_{c'}^T \mathbf{e}_i)} \right)$$

# Planetoid-T

Two phases

- 1 **pre-train**
- 2 training



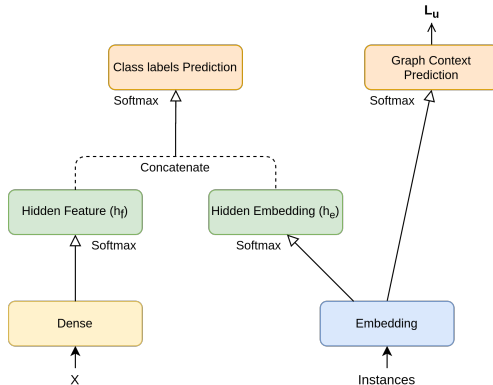
- Pretrained Embeddings leveraged during labels classification



# Planetoid-T

Two phases

- 1 pre-train
- 2 training

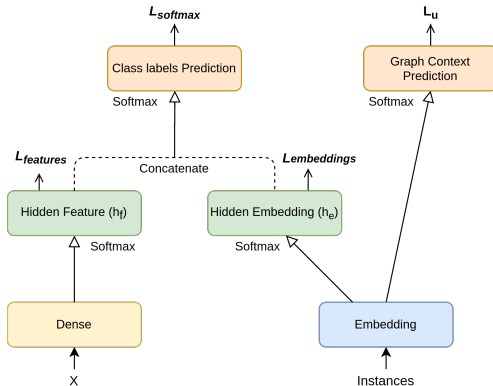


- Pretrained Embeddings leveraged during labels classification

# Planetoid-T

Two phases

- 1 pre-train
- 2 **training**

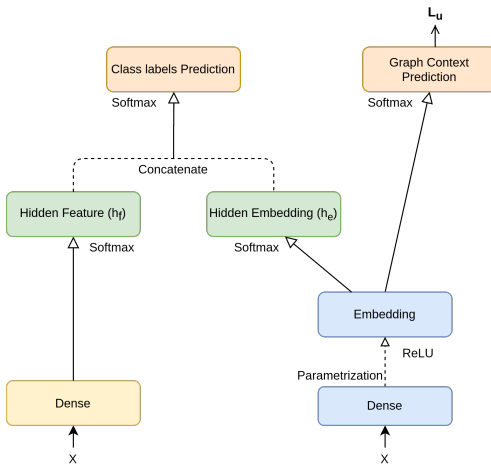


- Pretrained Embeddings leveraged during labels classification
- Total model loss:  $\mathcal{L} = \mathcal{L}_s = \mathcal{L}_{softmax} + \mathcal{L}_{features} + \mathcal{L}_{embeddings}$

# Planetoid-I

Two phases

- 1 pre-train
- 2 training

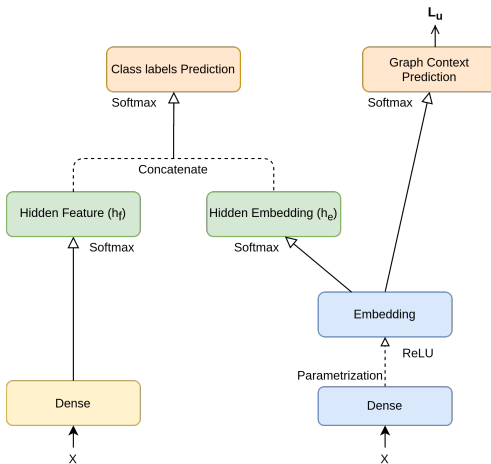


- Embeddings as a parameterized function of feature vectors

# Planetoid-I

Two phases

- 1 pre-train
- 2 training

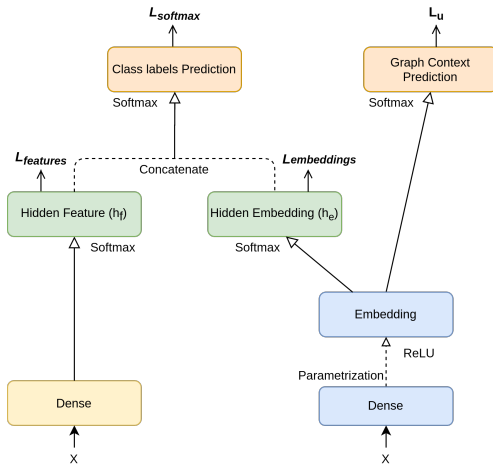


- Embeddings as a parameterized function of feature vectors

# Planetoid-I

Two phases

- 1 pre-train
- 2 training



- Embeddings as a parameterized function of feature vectors
- Total model loss:  $\mathcal{L} = \mathcal{L}_s + \lambda \mathcal{L}_u$

# Experimental Results

Method	Cora	Citeseer	Pubmed
<b>Planetoid</b>	75.7%	64.7%	77.2%
<b>Planetoid*</b>	73.1 $\pm$ 0.8%	62.3 $\pm$ 1.1%	73.7 $\pm$ 0.8%

\*: (our) 100 runs, Yang data split, *optimization std*

# Experimental Results

Method	Cora	Citeseer	Pubmed
Planetoid	75.7%	64.7%	77.2%
Planetoid*	$73.1 \pm 0.8\%$	$62.3 \pm 1.1\%$	$73.7 \pm 0.8\%$

## Performances Gap

- The authors validate the model on test set
- Our model evaluated on validation set and later on test set
- We report mean and std

\*: (our) 100 runs, Yang data split, *optimization std*

# Experimental Results

Method	Cora	Citeseer	Pubmed
Planetoid	75.7%	64.7%	77.2%
Planetoid*	$73.1 \pm 0.8\%$	$62.3 \pm 1.1\%$	$73.7 \pm 0.8\%$
Planetoid* (max)	75 %	66.2 %	75.5 %

## Performances Gap

- The authors validate the model on test set
- Our model evaluated on validation set and later on test set
- We report mean and std

\*: (our) 100 runs, Yang data split, *optimization std*



# Graph Attention Networks

(Veličković et al., 2018)

# GAT Contributions

## Anisotropy

- Attention to give **different importance** to **neighbors**
- Attention mechanism is **shared** to all nodes
  - no dependency on graph structure
  - graphs can be directed

# GAT Contributions

## Anisotropy

- Attention to give **different importance** to **neighbors**
- Attention mechanism is **shared** to all nodes
  - no dependency on graph structure
  - graphs can be directed

## Considerations

- No spectral convolution
- Sparse and parallel computations (high efficiency)

# Attention Layer

## Layer Configuration

- Input:  $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \dots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$
- Output:  $\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, \dots, \vec{h}'_N\}, \vec{h}'_i \in \mathbb{R}^{F'}$

# Attention Layer

## Layer Configuration

- Input:  $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \dots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$
- Output:  $\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, \dots, \vec{h}'_N\}, \vec{h}'_i \in \mathbb{R}^{F'}$

- 1 Normalized Attention Coefficients
- 2 Attention Layer Activation

# Attention Layer: Coefficients

For each node  $i$  and its neighborhood ( $\forall j \in \mathcal{N}_i$ ):

## Attention Coefficient

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

- $\mathbf{W} \in \mathbb{R}^{F' \times F}$  linear transformation
- Attentional mechanism parametrized by  $\mathbf{a} \in \mathbb{R}^{2F'}$

# Attention Layer: Coefficients

For each node  $i$  and its neighborhood ( $\forall j \in \mathcal{N}_i$ ):

## Attention Coefficient

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

- $\mathbf{W} \in \mathbb{R}^{F' \times F}$  linear transformation
- Attentional mechanism parametrized by  $\mathbf{a} \in \mathbb{R}^{2F'}$

## Normalized Attention Coefficient

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_k]\right)\right)}$$

# Attention Layer: Activation

## Normalized Attention Coefficient

$$\alpha_{ij} = \frac{\exp \left( \text{LeakyReLU} \left( \vec{\mathbf{a}}^T [\mathbf{W} \vec{h}_i \| \mathbf{W} \vec{h}_j] \right) \right)}{\sum_{k \in \mathcal{N}_i} \exp \left( \text{LeakyReLU} \left( \vec{\mathbf{a}}^T [\mathbf{W} \vec{h}_i \| \mathbf{W} \vec{h}_k] \right) \right)}$$



# Attention Layer: Activation

## Normalized Attention Coefficient

$$\alpha_{ij} = \frac{\exp \left( \text{LeakyReLU} \left( \vec{\mathbf{a}}^T [\mathbf{W} \vec{h}_i \| \mathbf{W} \vec{h}_j] \right) \right)}{\sum_{k \in \mathcal{N}_i} \exp \left( \text{LeakyReLU} \left( \vec{\mathbf{a}}^T [\mathbf{W} \vec{h}_i \| \mathbf{W} \vec{h}_k] \right) \right)}$$

$$\vec{h}'_i = \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j \right)$$

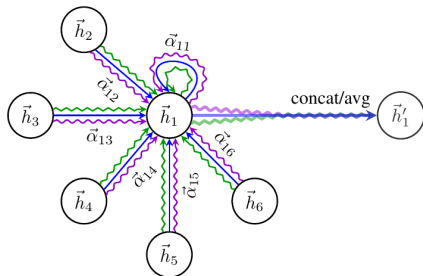
# Attention Layer: Activation

## Normalized Attention Coefficient

$$\alpha_{ij} = \frac{\exp \left( \text{LeakyReLU} \left( \vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_j] \right) \right)}{\sum_{k \in \mathcal{N}_i} \exp \left( \text{LeakyReLU} \left( \vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_k] \right) \right)}$$

$$\vec{h}'_i = \bigparallel_{k=1}^K \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j \right)$$

$$\vec{h}'_i = \sigma \left( \frac{1}{K} \sum_{k=1}^K \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j \right)$$



# GAT model

## Preprocessing

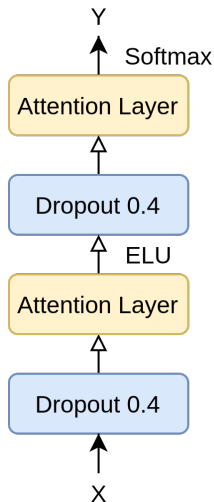
- row-normalize  $X$

# GAT model

Preprocessing

- row-normalize  $X$

Model



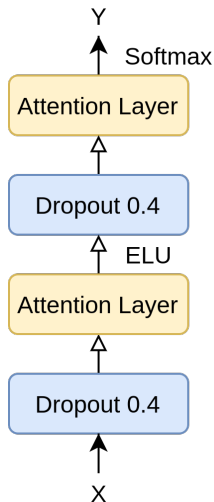
# GAT model

## Preprocessing

- row-normalize  $X$

## Model

- loss: *categorical cross-entropy*
- regularization: L2 on model weights
- hyperparameters: different heads numbers



# GAT model

## Preprocessing

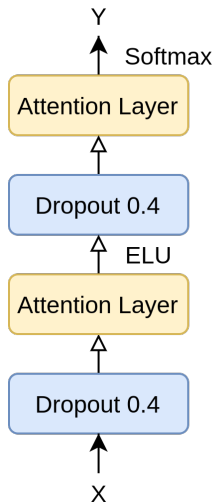
- row-normalize  $X$

## Model

- loss: *categorical cross-entropy*
- regularization: L2 on model weights
- hyperparameters: different heads numbers

## Evaluation

- metric: classification accuracy on test set



# Experimental Results

Method	Cora	Citeseer	Pubmed
<b>GAT</b>	$83.0 \pm 0.7\%$	$72.5 \pm 0.7\%$	$79.0 \pm 0.3\%$
<b>GAT*</b>	$83.1 \pm 0.4\%$	$71.7 \pm 0.7\%$	$77.7 \pm 0.4\%$

\*: (our) 100 runs, Yang data split, *optimization std*

# Conclusions



# Experimental Results

Method	Cora	Citeseer	Pubmed
<b>Planetoid</b>	75.7%	64.7%	77.2%
<b>Planetoid*</b>	73.1 $\pm$ 0.9%	62.3 $\pm$ 1.1%	73.6 $\pm$ 0.7%
<b>Planetoid**</b>	72.3 $\pm$ 2.0%	59.4 $\pm$ 2.0%	69.7 $\pm$ 2.8%

\*: (our) 100 runs, Yang data split, *optimization std*

\*\*: (our) 100 runs, fixed model seed, *data split std*

# Experimental Results

Method	Cora	Citeseer	Pubmed
<b>Planetoid</b>	75.7%	64.7%	77.2%
<b>Planetoid*</b>	73.1 $\pm$ 0.9%	62.3 $\pm$ 1.1%	73.6 $\pm$ 0.7%
<b>Planetoid**</b>	72.3 $\pm$ 2.0%	59.4 $\pm$ 2.0%	69.7 $\pm$ 2.8%
<b>ChebNet</b>	81.2%	69.8%	74.4%
<b>ChebNet*</b>	82.0 $\pm$ 0.6%	70.5 $\pm$ 0.7%	75.2 $\pm$ 1.8%
<b>ChebNet**</b>	78.9 $\pm$ 1.8%	68.2 $\pm$ 1.9%	73.4 $\pm$ 2.4%

\*: (our) 100 runs, Yang data split, *optimization std*

\*\* : (our) 100 runs, fixed model seed, *data split std*

# Experimental Results

Method	Cora	Citeseer	Pubmed
<b>Planetoid</b>	75.7%	64.7%	77.2%
<b>Planetoid*</b>	73.1 $\pm$ 0.9%	62.3 $\pm$ 1.1%	73.6 $\pm$ 0.7%
<b>Planetoid**</b>	72.3 $\pm$ 2.0%	59.4 $\pm$ 2.0%	69.7 $\pm$ 2.8%
<b>ChebNet</b>	81.2%	69.8%	74.4%
<b>ChebNet*</b>	82.0 $\pm$ 0.6%	70.5 $\pm$ 0.7%	75.2 $\pm$ 1.8%
<b>ChebNet**</b>	78.9 $\pm$ 1.8%	68.2 $\pm$ 1.9%	73.4 $\pm$ 2.4%
<b>GCN</b>	81.5%	70.3%	79.0%
<b>GCN*</b>	80.6 $\pm$ 0.6%	68.7 $\pm$ 0.9%	78.3 $\pm$ 0.5%
<b>GCN**</b>	79.2 $\pm$ 1.7%	68.0 $\pm$ 1.8%	76.2 $\pm$ 2.5%
<b>GAT</b>	83.0 $\pm$ 0.7%	72.5 $\pm$ 0.7%	79.0 $\pm$ 0.3%
<b>GAT*</b>	83.1 $\pm$ 0.4%	71.7 $\pm$ 0.7%	77.7 $\pm$ 0.4%
<b>GAT**</b>	81.0 $\pm$ 1.7%	69.7 $\pm$ 1.7%	77.4 $\pm$ 2.4%

\*: (our) 100 runs, Yang data split, *optimization std*

\*\* : (our) 100 runs, fixed model seed, *data split std*

# Summary

## Objective

Compare performances of **Planetoid**, **ChebNet**, **GCN**, **GAT** on Cora, Citeseer and Pubmed

# Summary

## Objective

Compare performances of **Planetoid**, **ChebNet**, **GCN**, **GAT** on Cora, Citeseer and Pubmed

## Final Considerations

- Our models reach comparable results w.r.t. the proposed ones
- Higher std with different data splits

# Summary

## Objective

Compare performances of **Planetoid**, **ChebNet**, **GCN**, **GAT** on Cora, Citeseer and Pubmed

## Final Considerations

- Our models reach comparable results w.r.t. the proposed ones
- Higher std with different data splits

## Next

- Better benchmark framework for GNNs
- More realistic data split

Thanks for the attention

# References I



Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, 2016.



Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson, *Benchmarking graph neural networks*, 2020.



P. Frasconi, M. Gori, and A. Sperduti, *A general framework for adaptive processing of data structures*, IEEE Transactions on Neural Networks **9** (1998), no. 5, 768–786.








C. Giles, Kurt Bollacker, and Steve Lawrence, *Citeseer: An automatic citation indexing system*, Proceedings of 3rd ACM Conference on Digital Libraries (2000).



M. Gori, G. Monfardini, and F. Scarselli, *A new model for learning in graph domains*, Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., vol. 2, 2005, pp. 729–734 vol. 2.



# References II

-  Thomas N. Kipf and Max Welling, *Semi-supervised classification with graph convolutional networks*, 2016.
-  A.K. McCallum, *Automating the construction of internet portals with machine learning*, Information Retrieval **3** (2000), 127–163.
-  Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio, *Graph attention networks*, 2017.
-  Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu, *A comprehensive survey on graph neural networks*, IEEE Transactions on Neural Networks and Learning Systems (2020), 1–21.
-  Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov, *Revisiting semi-supervised learning with graph embeddings*, 2016.