Convolutional Graph Embeddings

Si Yi (Cathy) Meng

March 11, 2019

UBC MLRG

Motivation

- Graphs are everywhere.
- Machine Learning tasks on graphs:
    - Node classification
    - Link prediction
    - Neighbourhood identification
    - ...
- **Representation learning on graphs**: learn vector rerpresentations of nodes or subgraphs for downstream ML tasks.

Motivation

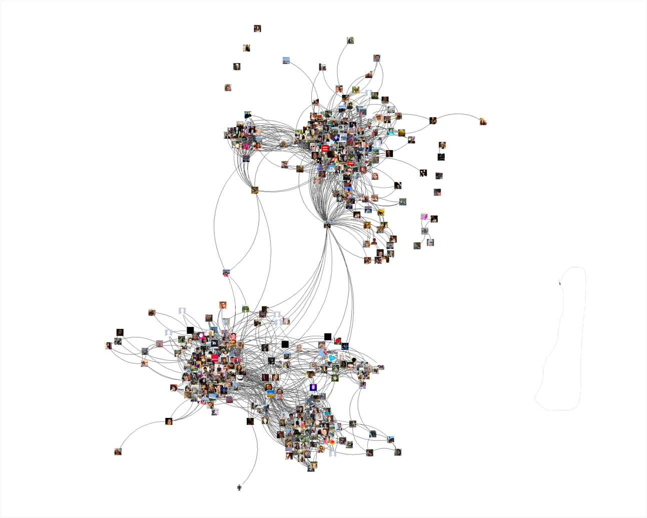


Figure 1: Facebook friendship network

Motivation

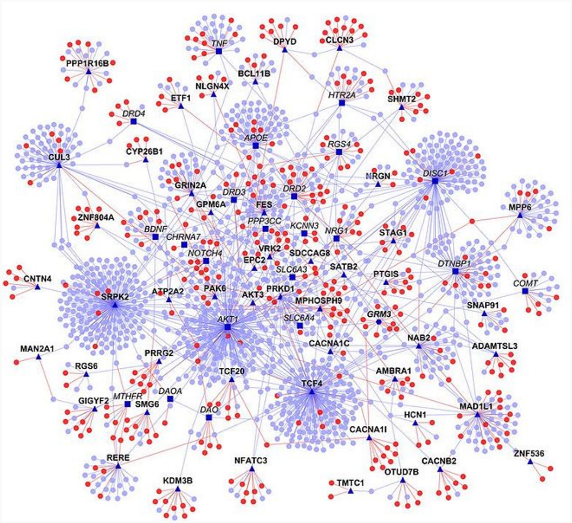


Figure 2: Schizophrenia PPIs

Table of contents

Node Embeddings

Node Embeddings

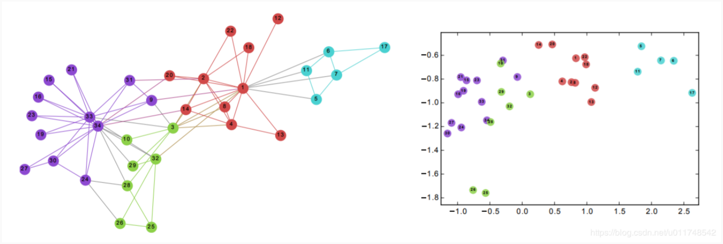


Figure 3: Perozzi et al. 2014. [6]

- The vector representation of nodes should preserve information about pairwise relationships.

Node Embeddings

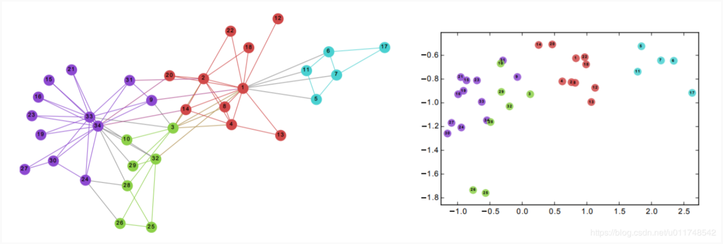


Figure 3: Perozzi et al. 2014. [6]

- The vector representation of nodes should preserve information about pairwise relationships.
- But mapping from non-Euclidean space to a feature vector is not straightforward.

Notation

- Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $|\mathcal{V}| = n$ nodes

Notation

- Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $|\mathcal{V}| = n$ nodes
- Adjacency matrix $A \in \mathbb{R}^{n \times n}$, binary or weighted

Notation

- Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $|\mathcal{V}| = n$ nodes
- Adjacency matrix $A \in \mathbb{R}^{n \times n}$, binary or weighted
- Degree matrix D where $D_{ii} = \sum_j A_{ij}$, diagonal

Notation

- Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $|\mathcal{V}| = n$ nodes
- Adjacency matrix $A \in \mathbb{R}^{n \times n}$, binary or weighted
- Degree matrix D where $D_{ii} = \sum_j A_{ij}$, diagonal
- May also have node attributes $X \in \mathbb{R}^{n \times d}$

Node Embeddings - Shallow Embeddings

Encoder-decoder framework

$\texttt{ENC}(v_i) = Zv_i$

$\quad Z \in \mathbb{R}^{m \times n},\ v_i = \text{one-hot}$

Node Embeddings - Shallow Embeddings

Encoder-decoder framework

$\texttt{ENC}(v_i) = Zv_i$

$Z \in \mathbb{R}^{m \times n}$, $v_i =$ one-hot

$\texttt{DEC}(\texttt{ENC}(v_i), \texttt{ENC}(v_j)) = \texttt{DEC}(z_i, z_j) \approx s_{\mathcal{G}}(v_i, v_j)$

where $s_{\mathcal{G}}$ is a pre-defined similarity metric between two nodes, defined over the graph.

Node Embeddings - Shallow Embeddings

- Matrix factorization-based approaches:

Node Embeddings - Shallow Embeddings

- Matrix factorization-based approaches:
    - Deterministic similarity measure such as A_{ij}.

Node Embeddings - Shallow Embeddings

- Matrix factorization-based approaches:
    - Deterministic similarity measure such as A_{ij}.
    - Minimize the reconstruction error: $\mathcal{L} = \sum_{i,j} \ell\left(\texttt{DEC}(z_i, z_j), s_{\mathcal{G}}(v_i, v_j)\right) \approx \|Z^T Z - S\|_2^2$.

Node Embeddings - Shallow Embeddings

- Matrix factorization-based approaches:
    - Deterministic similarity measure such as A_{ij}.
    - Minimize the reconstruction error: $\mathcal{L} = \sum_{i,j} \ell\left(\mathtt{DEC}(z_i, z_j), s_{\mathcal{G}}(v_i, v_j)\right) \approx \|Z^T Z - S\|_2^2$.
- Random walk approaches:

Node Embeddings - Shallow Embeddings

- Matrix factorization-based approaches:
    - Deterministic similarity measure such as A_{ij}.
    - Minimize the reconstruction error: $\mathcal{L} = \sum_{i,j} \ell\left(\texttt{DEC}(z_i, z_j), s_{\mathcal{G}}(v_i, v_j)\right) \approx \|Z^T Z - S\|_2^2$.
- Random walk approaches:
    - Stochastic measure of node similarity based on random walk statistics.

Node Embeddings - Shallow Embeddings

- Matrix factorization-based approaches:
    - Deterministic similarity measure such as A_{ij}.
    - Minimize the reconstruction error: $\mathcal{L} = \sum_{i,j} \ell\left(\mathtt{DEC}(z_i, z_j), s_{\mathcal{G}}(v_i, v_j)\right) \approx \|Z^T Z - S\|_2^2$.
- Random walk approaches:
    - Stochastic measure of node similarity based on random walk statistics.
    - Decoder uses softmax over the inner products of the encoded features.

See Hamilton et al. 2017 [3] for an in-depth review.

Node Embeddings - Shallow Embeddings

- These approaches are simple and intuitive.

Node Embeddings - Shallow Embeddings

- These approaches are simple and intuitive.
- But . . .
    - No parameter sharing as the encoder is just a lookup table.

Node Embeddings - Shallow Embeddings

- These approaches are simple and intuitive.
- But ...
    - No parameter sharing as the encoder is just a lookup table.
        - Overfitting
        - Very costly as for large n
    - Not utilizing node features X.

Node Embeddings - Shallow Embeddings

- These approaches are simple and intuitive.
- But ...
    - No parameter sharing as the encoder is just a lookup table.
        - Overfitting
        - Very costly as for large n
    - Not utilizing node features X.
    - Inherently transductive.

Node Embeddings - Shallow Embeddings

- These approaches are simple and intuitive.
- But . . .
    - No parameter sharing as the encoder is just a lookup table.
        - Overfitting
        - Very costly as for large n
    - Not utilizing node features X.
    - Inherently transductive.
- Instead of a lookup table, we could use a neural network to encode a node's local structure.

Node Embeddings - Shallow Embeddings

- These approaches are simple and intuitive.
- But . . .
    - No parameter sharing as the encoder is just a lookup table.
        - Overfitting
        - Very costly as for large n
    - Not utilizing node features X.
    - Inherently transductive.
- Instead of a lookup table, we could use a neural network to encode a node's local structure.
- We will focus on methods using convolution operations on graphs.

Convolution on Graphs (Graph-CNN)

Convolution on Graphs - a Spectral Formulation

How do we define localized convolutional filters on graphs?

We don't have grids or sequences to define a fixed-size neighborhood.

the Graph Laplacian

Unormalized graph Laplacian $\Delta = D - A$

- Symmetric normalized graph Laplacian
  $L := D^{-1/2}\Delta D^{-1/2} = I_n - D^{-1/2}AD^{-1/2}$
- $L = L^T \succeq 0$
- Multiplicity of the eigenvalue 0 indicates the number of connected components in the graph.

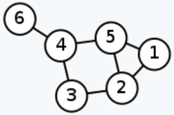
Labeled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

Figure 4: Example of Laplacian Matrix

the Graph Laplacian

Since $L = L^T \succeq 0$, for $\ell \in [1, n]$, it has

- Complete set of orthonormal eigenvectors $\{u_\ell\} \in \mathbb{R}^n$

the Graph Laplacian

Since $L = L^T \succeq 0$, for $\ell \in [1, n]$, it has

- Complete set of orthonormal eigenvectors $\{u_\ell\} \in \mathbb{R}^n$
  - also called "Fourier modes", "Fourier basis functions"
  - related to spectral clustering

the Graph Laplacian

Since $L = L^T \succeq 0$, for $\ell \in [1, n]$, it has

- Complete set of orthonormal eigenvectors $\{u_\ell\} \in \mathbb{R}^n$
    - also called "Fourier modes", "Fourier basis functions"
    - related to spectral clustering
- Corresponding ordered real nonnegative eigenvalues $\{\lambda_\ell\}$

the Graph Laplacian

Since $L = L^T \succeq 0$, for $\ell \in [1, n]$, it has

- Complete set of orthonormal eigenvectors $\{u_\ell\} \in \mathbb{R}^n$
    - also called "Fourier modes", "Fourier basis functions"
    - related to spectral clustering
- Corresponding ordered real nonnegative eigenvalues $\{\lambda_\ell\}$
    - "frequencies of the graph"

the Graph Laplacian

Since $L = L^T \succeq 0$, for $\ell \in [1, n]$, it has

- Complete set of orthonormal eigenvectors $\{u_\ell\} \in \mathbb{R}^n$
    - also called "Fourier modes", "Fourier basis functions"
    - related to spectral clustering
- Corresponding ordered real nonnegative eigenvalues $\{\lambda_\ell\}$
    - "frequencies of the graph"
- $L = U \Lambda U^T$, $U = [u_1, \ldots, u_n]$, $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$

Spectral Filtering

Let $x \in \mathbb{R}^n$ be a signal vector for all the nodes (we can generalize this to a vector per node). The graph Fourier transform of x is defined as

$$\hat{x} = U^T x$$

and the inverse GFT is is $x = U\hat{x}$.

Spectral Filtering

Let $x \in \mathbb{R}^n$ be a signal vector for all the nodes (we can generalize this to a vector per node). The graph Fourier transform of x is defined as

$$\hat{x} = U^T x$$

and the inverse GFT is is $x = U\hat{x}$.
Define the spectral convolution as the multiplication of a signal with a filter $g_\theta = \text{diag}(\theta)$ parameterized by coefficients $\theta \in \mathbb{R}^n$ in the Fourier domain as

$$g_\theta \star x = U g_\theta U^T x$$

GFT of x, apply filter in Fourier domain, then transform back. Note that g_θ is a function of Λ.

Spectral Filtering

- But since g_θ is non-parametric, learning is expensive.

Spectral Filtering

- But since g_θ is non-parametric, learning is expensive.
- And it's not localized in space.

Spectral Filtering

- But since g_θ is non-parametric, learning is expensive.
- And it's not localized in space.

Solution: Approximate $g_\theta(\Lambda)$ with a polynomial filter

$$g_\theta(\Lambda) \approx \sum_{k=0}^{K-1} \theta_k \Lambda^k,$$

where $\theta = [\theta_0, \ldots, \theta_{K-1}]$ is now of size independent of n.

Spectral Filtering

- But computing the eigendecomposition of L is also expensive for large graphs

Spectral Filtering

- But computing the eigendecomposition of L is also expensive for large graphs

Solution: Approximate $g_\theta(\Lambda)$ with a K^{th}-order truncated expansion of Chebyshev polynomials $T_k(x)$:

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

with a rescaled $\hat{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - \boldsymbol{I_n}$.

The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$.

Spectral Filtering

The filtering operation $g_\theta \star x$ can now be written as

$$\begin{aligned} g_\theta \star x &= U g_\theta U^T x \\ &\approx U\Big(\sum_{k=0}^{K-1} \theta_k T_k(\hat{\Lambda})\Big) U^T x \end{aligned}$$

Spectral Filtering

The filtering operation $g_\theta \star x$ can now be written as

$$\begin{aligned} g_\theta \star x &= U g_\theta U^T x \\ &\approx U\Big(\sum_{k=0}^{K-1} \theta_k T_k(\hat{\Lambda})\Big) U^T x \\ &= \sum_{k=0}^{K-1} \theta_k T_k(\hat{L}) x \end{aligned}$$

Spectral Filtering

The filtering operation $g_\theta \star x$ can now be written as

$$\begin{aligned} g_\theta \star x &= U g_\theta U^T x \\ &\approx U \Big(\sum_{k=0}^{K-1} \theta_k T_k(\hat{\Lambda}) \Big) U^T x \\ &= \sum_{k=0}^{K-1} \theta_k T_k(\hat{L}) x \end{aligned}$$

with $\hat{L} = \frac{2}{\lambda_{\max}} L - I_n$, and the last equality comes from $L^k = (U \Lambda U^T)^k = U \Lambda^k U^T$.

Spectral Filtering

The spectral filter represented by L is also localized:

- It can be shown that $d_{\mathcal{G}}(i,j) > k' \implies (L^{k'})_{i,j} = 0$, where $d_{\mathcal{G}}(i,j)$ is the shortest path distance between two vertices.

Spectral Filtering

The spectral filter represented by L is also localized:

- It can be shown that $d_{\mathcal{G}}(i,j) > k' \implies (L^{k'})_{i,j} = 0$, where $d_{\mathcal{G}}(i,j)$ is the shortest path distance between two vertices.
- g_θ operates on the K-hop neighbors of a vertex!

Now we got rid of the eigendecomposition.

Spectral Filtering

The spectral filter represented by L is also localized:

- It can be shown that $d_{\mathcal{G}}(i,j) > k' \implies (L^{k'})_{i,j} = 0$, where $d_{\mathcal{G}}(i,j)$ is the shortest path distance between two vertices.
- g_θ operates on the K-hop neighbors of a vertex!

Now we got rid of the eigendecomposition.

So what's the algorithm?

Chebyshev Spectral Graph Convolution

We had a feature matrix $X \in \mathbb{R}^{n \times d}$, let $H_k = T_k(\hat{L})X \in \mathbb{R}^{n \times d}$, then we have

$$\begin{aligned} H_0 &= X \\ H_1 &= \hat{L}X \\ H_k &= 2\hat{L}H_{k-1} - H_{k-2} \end{aligned}$$

Chebyshev Spectral Graph Convolution

We had a feature matrix $X \in \mathbb{R}^{n \times d}$, let $H_k = T_k(\hat{L})X \in \mathbb{R}^{n \times d}$, then we have

$$\begin{aligned} H_0 &= X \\ H_1 &= \hat{L}X \\ H_k &= 2\hat{L}H_{k-1} - H_{k-2} \end{aligned}$$

The filtering operation costs $O(K|\mathcal{E}|)$, and the corresponding K-hop convolution operation is

$$X' = \sum_{k=0}^{K-1} H_k \Theta_k,$$

where $\Theta_k \in \mathbb{R}^{d \times m}$ for a desired output size m. We can now use X' as a feature extractor (node embeddings).

Graph-CNN

Remarks:

- Convolution followed by non-linear activation.

Graph-CNN

Remarks:

- Convolution followed by non-linear activation.
- Graph coarsening/downsampling to group together similar vertices.

Graph-CNN

Remarks:

- Convolution followed by non-linear activation.
- Graph coarsening/downsampling to group together similar vertices.
    - Graph clustering, but NP-hard.
    - Greedy algorithm: Graclus multilevel clustering, gives successive coarsened graphs.

Graph-CNN

Remarks:

- Convolution followed by non-linear activation.
- Graph coarsening/downsampling to group together similar vertices.
    - Graph clustering, but NP-hard.
    - Greedy algorithm: Graclus multilevel clustering, gives successive coarsened graphs.
- Graph pooling: create balanced binary tree to remember which nodes were matched to perform pooling.
- For more information, see [1, 4].

Graph-CNN

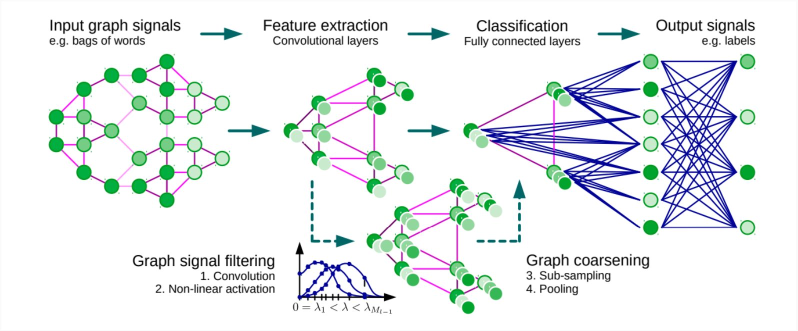


Figure 5: Architecture of a CNN on graphs and the four ingredients of a (graph) convolutional layer. Defferrard et al. 2016 [1].

Graph Convolutional Networks (GCN)

GCN

Kipf & Welling [5] introduced the multi-layer Graph Convolutional Network (GCN) with the following layer-wise propagation rule:

$$H_{\ell+1} = \sigma\big(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H_\ell W_\ell\big), \tag{1}$$

where $\tilde{A} = A + I_n$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-loops, $\sigma(\cdot)$ is some nonlinear activation, and $H_0 = X$.

GCN

Recall the Chebyshev spectral graph convolution derived earlier,

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

For $K = 2$ and approximate $\lambda_{\max} \approx 2$, we have

GCN

Recall the Chebyshev spectral graph convolution derived earlier,

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

For $K = 2$ and approximate $\lambda_{\max} \approx 2$, we have

$$g_\theta \star x \approx \theta_0 x + \theta_1 \hat{L} x$$

GCN

Recall the Chebyshev spectral graph convolution derived earlier,

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

For $K = 2$ and approximate $\lambda_{\max} \approx 2$, we have

$$\begin{aligned} g_\theta \star x &\approx \theta_0 x + \theta_1 \hat{L} x \\ &= \theta_0 x + \theta_1 (\frac{2}{\lambda_{\max}} L - I_n) x \end{aligned}$$

GCN

Recall the Chebyshev spectral graph convolution derived earlier,

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

For $K = 2$ and approximate $\lambda_{\max} \approx 2$, we have

$$\begin{aligned} g_\theta \star x &\approx \theta_0 x + \theta_1 \hat{L}x \\ &= \theta_0 x + \theta_1 (\frac{2}{\lambda_{\max}} L - I_n)x \\ &\approx \theta_0 x + \theta_1 (L - I_n)x \end{aligned}$$

GCN

Recall the Chebyshev spectral graph convolution derived earlier,

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

For $K = 2$ and approximate $\lambda_{\max} \approx 2$, we have

$$\begin{aligned}
g_\theta \star x &\approx \theta_0 x + \theta_1 \hat{L} x \\
&= \theta_0 x + \theta_1 (\frac{2}{\lambda_{\max}} L - I_n) x \\
&\approx \theta_0 x + \theta_1 (L - I_n) x \\
&= \theta_0 x - \theta_1 D^{-1/2} A D^{-1/2} x
\end{aligned}$$

GCN

Recall the Chebyshev spectral graph convolution derived earlier,

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

For $K = 2$ and approximate $\lambda_{\max} \approx 2$, we have

$$\begin{aligned} g_\theta \star x &\approx \theta_0 x + \theta_1 \hat{L} x \\ &= \theta_0 x + \theta_1 (\frac{2}{\lambda_{\max}} L - I_n) x \\ &\approx \theta_0 x + \theta_1 (L - I_n) x \\ &= \theta_0 x - \theta_1 D^{-1/2} A D^{-1/2} x \\ &= \theta (I_n + D^{-1/2} A D^{-1/2}) x && \text{By letting } \theta = \theta_0 = -\theta_1 \end{aligned}$$

The eigenvalues of $I_n + D^{-1/2} A D^{-1/2}$ are in range $[0, 2]$, which may lead to numerical instability in repeated applications of this filter.

GCN

Renormalization trick:

$$I_n + D^{-1/2} A D^{-1/2} \rightarrow \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$$

Generalizing to node signals of multiple dimensions and using W as the parameters instead of θ, we get the convolution operation (prior to activation) in eq.1,

$$Z = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X W$$

where $W \in \mathbb{R}^{d \times m}$ for a desired output size m.

GCN

Renormalization trick:

$$I_n + D^{-1/2} A D^{-1/2} \rightarrow \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$$

Generalizing to node signals of multiple dimensions and using W as the parameters instead of θ, we get the convolution operation (prior to activation) in eq.1,

$$Z = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X W$$

where $W \in \mathbb{R}^{d \times m}$ for a desired output size m.
The cost of the filtering operation (prior to multiplication by W) is $O(|\mathcal{E}|)$, and and all matrix multiplications here can be efficiently computed.

GCN - Semi-supervised Classification

To perform semi-supervised classification under this framework, first compute $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$. The 2-layer forward model used is

$$\hat{Y} = \text{softmax}(\hat{A}\,\text{ReLU}(\hat{A}XW_0)W_1)$$

The cross-entropy loss is applied over all labeled examples

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}_L} \sum_{c=1}^{K} Y_{ic} \ln \hat{Y}_{ic}$$

where $\mathcal{Y}_L$ is the set of node indices where labels exist.

GCN

Remarks:

- No longer limited to the explicit parameterization given by the Chebyshev polynomials.

GCN

Remarks:

- No longer limited to the explicit parameterization given by the Chebyshev polynomials.
- Alleviate the problem of overfitting on local neighborhood structures for graphs with very wide node degree distributions.

GCN

Remarks:

- No longer limited to the explicit parameterization given by the Chebyshev polynomials.
- Alleviate the problem of overfitting on local neighborhood structures for graphs with very wide node degree distributions.
- Scalable.

GCN

Remarks:

- No longer limited to the explicit parameterization given by the Chebyshev polynomials.
- Alleviate the problem of overfitting on local neighborhood structures for graphs with very wide node degree distributions.
- Scalable.
- But it is transductive in nature.

Inductive Representation Learning on Large Graphs (GraphSAGE)

GraphSAGE

Goal: Efficiently generate node embeddings for nodes unseen at training time, or entirely new graphs.

- Essential for high throughput, production level systems.

GraphSAGE

Goal: Efficiently generate node embeddings for nodes unseen at training time, or entirely new graphs.

- Essential for high throughput, production level systems.
- Generalization across graphs with similar structures.

GraphSAGE

Goal: Efficiently generate node embeddings for nodes unseen at training time, or entirely new graphs.

- Essential for high throughput, production level systems.
- Generalization across graphs with similar structures.
- How to achieve this without re-training with the entire graph?

GraphSAGE

GraphSAGE: Sample and Aggregate (Hamilton et al. [2])

- Train a set of *aggregator functions* that learn to aggregate feature information from a node's local neighborhood.

GraphSAGE

GraphSAGE: Sample and Aggregate (Hamilton et al. [2])

- Train a set of *aggregator functions* that learn to aggregate feature information from a node's local neighborhood.
    - Learn how to aggregate node features, degree statistics, etc.
- At test time, apply the learned aggregation functions to generate embeddings for entirely unseen nodes.

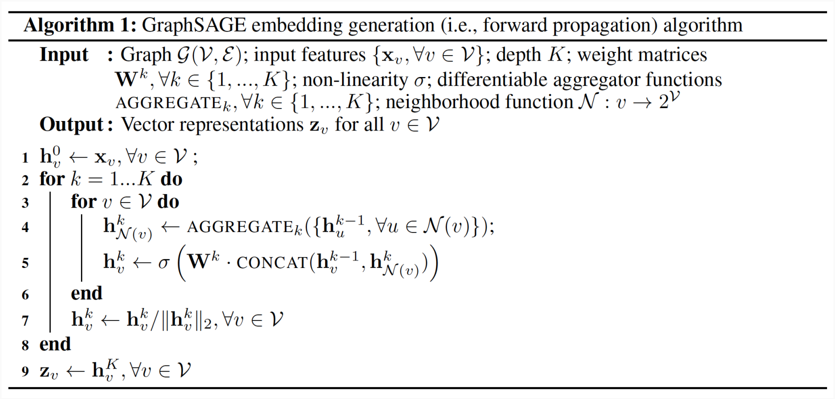
GraphSAGE

GraphSAGE: Sample and Aggregate (Hamilton et al. [2])

- Train a set of *aggregator functions* that learn to aggregate feature information from a node's local neighborhood.
    - Learn how to aggregate node features, degree statistics, etc.
- At test time, apply the learned aggregation functions to generate embeddings for entirely unseen nodes.
- Unsupervised loss function.

GraphSAGE - Embedding Generation

Assume $\forall k \in \{1, \dots, K\}$, the $\texttt{AGGREGATE}_k$ functions are learned, as well as a set of weights W_k, the embedding generation procedure is



Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

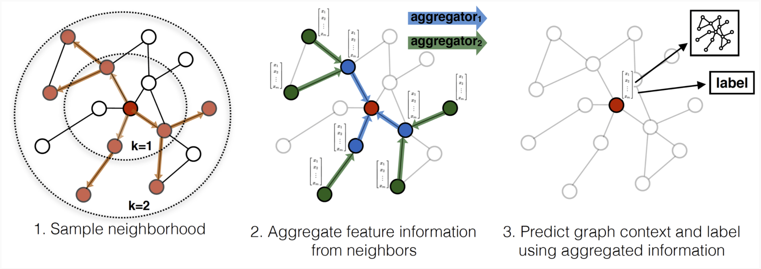
Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

```
$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
for $k = 1...K$ do
    for $v \in \mathcal{V}$ do
        $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
        $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
    end
    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

Figure 6: Hamilton et al. [2]

GraphSAGE - Embedding Generation



k=1
k=2
1. Sample neighborhood
aggregator1
aggregator2
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Figure 7: Hamilton et al. [2]

After K iterations, each node's embedding will contain information for all its K-hop neighbors. In the minibatch setting, first forward sample the required neighborhood sets and then run the inner loop.

GraphSAGE

- Uniformly sample a fixed-size set of neighbors to keep the computional cost of each batch under control.

GraphSAGE

- Uniformly sample a fixed-size set of neighbors to keep the computional cost of each batch under control.
- Graph-based loss function (unsupervised):

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log(\sigma(\mathbf{z}_u^T \mathbf{z}_v)) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-\mathbf{z}_u^T \mathbf{z}_{v_n}))$$

  - v: a node that co-occurs near u on a fixed-length random walk
  - σ: sigmoid
  - P_n: negative sampling distribution
  - Q: number of negative samples

GraphSAGE

- Uniformly sample a fixed-size set of neighbors to keep the computional cost of each batch under control.
- Graph-based loss function (unsupervised):

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log(\sigma(\mathbf{z}_u^T \mathbf{z}_v)) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-\mathbf{z}_u^T \mathbf{z}_{v_n}))$$

  - v: a node that co-occurs near u on a fixed-length random walk
  - σ: sigmoid
  - P_n: negative sampling distribution
  - Q: number of negative samples
- Can also replace/augment this loss with a supervised, task-specific objective.

GraphSAGE - Aggregator Functions

- **Mean aggregator**
    - Elementwise mean of $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$.

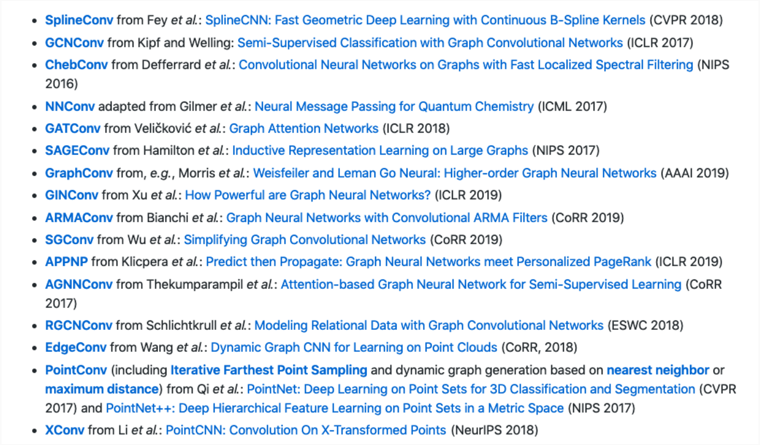
GraphSAGE - Aggregator Functions

- **Mean aggregator**
    - Elementwise mean of $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$.
- **LSTM aggregator**
    - LSTMs operate on sequences.
    - Apply LSTMs to a random permutation of a node's neighbors.

GraphSAGE - Aggregator Functions

- **Mean aggregator**
  - Elementwise mean of $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$.
- **LSTM aggregator**
  - LSTMs operate on sequences.
  - Apply LSTMs to a random permutation of a node's neighbors.
- **Pooling aggregator**
  - Each neighbor's vector is independently fed through a FC layer.
  - Then perform elementwise max-pooling.
  - $\texttt{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}), \forall u_i \in \mathcal{N}(v)\})$

Other methods



- **SplineConv** from Fey *et al.*: SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels (CVPR 2018)
- **GCNConv** from Kipf and Welling: Semi-Supervised Classification with Graph Convolutional Networks (ICLR 2017)
- **ChebConv** from Defferrard *et al.*: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (NIPS 2016)
- **NNConv** adapted from Gilmer *et al.*: Neural Message Passing for Quantum Chemistry (ICML 2017)
- **GATConv** from Veličković *et al.*: Graph Attention Networks (ICLR 2018)
- **SAGEConv** from Hamilton *et al.*: Inductive Representation Learning on Large Graphs (NIPS 2017)
- **GraphConv** from, *e.g.*, Morris *et al.*: Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks (AAAI 2019)
- **GINConv** from Xu *et al.*: How Powerful are Graph Neural Networks? (ICLR 2019)
- **ARMAConv** from Bianchi *et al.*: Graph Neural Networks with Convolutional ARMA Filters (CoRR 2019)
- **SGConv** from Wu *et al.*: Simplifying Graph Convolutional Networks (CoRR 2019)
- **APPNP** from Klicpera *et al.*: Predict then Propagate: Graph Neural Networks meet Personalized PageRank (ICLR 2019)
- **AGNNConv** from Thekumparampil *et al.*: Attention-based Graph Neural Network for Semi-Supervised Learning (CoRR 2017)
- **RGCNConv** from Schlichtkrull *et al.*: Modeling Relational Data with Graph Convolutional Networks (ESWC 2018)
- **EdgeConv** from Wang *et al.*: Dynamic Graph CNN for Learning on Point Clouds (CoRR, 2018)
- **PointConv** (including **Iterative Farthest Point Sampling** and dynamic graph generation based on **nearest neighbor** or **maximum distance**) from Qi *et al.*: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation (CVPR 2017) and PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space (NIPS 2017)
- **XConv** from Li *et al.*: PointCNN: Convolution On X-Transformed Points (NeurIPS 2018)

Figure 8: PyTorch geometric

Libraries

Deep Graph Library (DGL)

build passing license Apache 2.0

Figure 9: DGL

Figure 10: PyTorch geometric

References i

M. Defferrard, X. Bresson, and P. Vandergheynst.
Convolutional neural networks on graphs with fast localized spectral filtering.
In *Advances in neural information processing systems*, pages 3844–3852, 2016.

W. Hamilton, Z. Ying, and J. Leskovec.
Inductive representation learning on large graphs.
In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

W. L. Hamilton, R. Ying, and J. Leskovec.
Representation learning on graphs: Methods and applications.
arXiv preprint arXiv:1709.05584, 2017.

References ii

D. K. Hammond, P. Vandergheynst, and R. Gribonval.
Wavelets on graphs via spectral graph theory.
Applied and Computational Harmonic Analysis, 30(2):129–150, 2011.

T. N. Kipf and M. Welling.
Semi-supervised classification with graph convolutional networks.
arXiv preprint arXiv:1609.02907, 2016.

B. Perozzi, R. Al-Rfou, and S. Skiena.
Deepwalk: Online learning of social representations.
In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.